2

DTNSRDC/CMLD-85/02

DAVID W. TAYLOR NAVAL SHIP RESEARCH AND DEVELOPMENT CENTER

Bethesda, Maryland 20084

ZOG/VINSON TECHNOLOGY DEMONSTRATION PROJECT

SYSTEM DESCRIPTION

VOLUME I

by

Donald J. Schmelter
Ron Lupish

APPROVED FOR PUBLIC RELEASE: DISTRIBUTION UNLIMITED

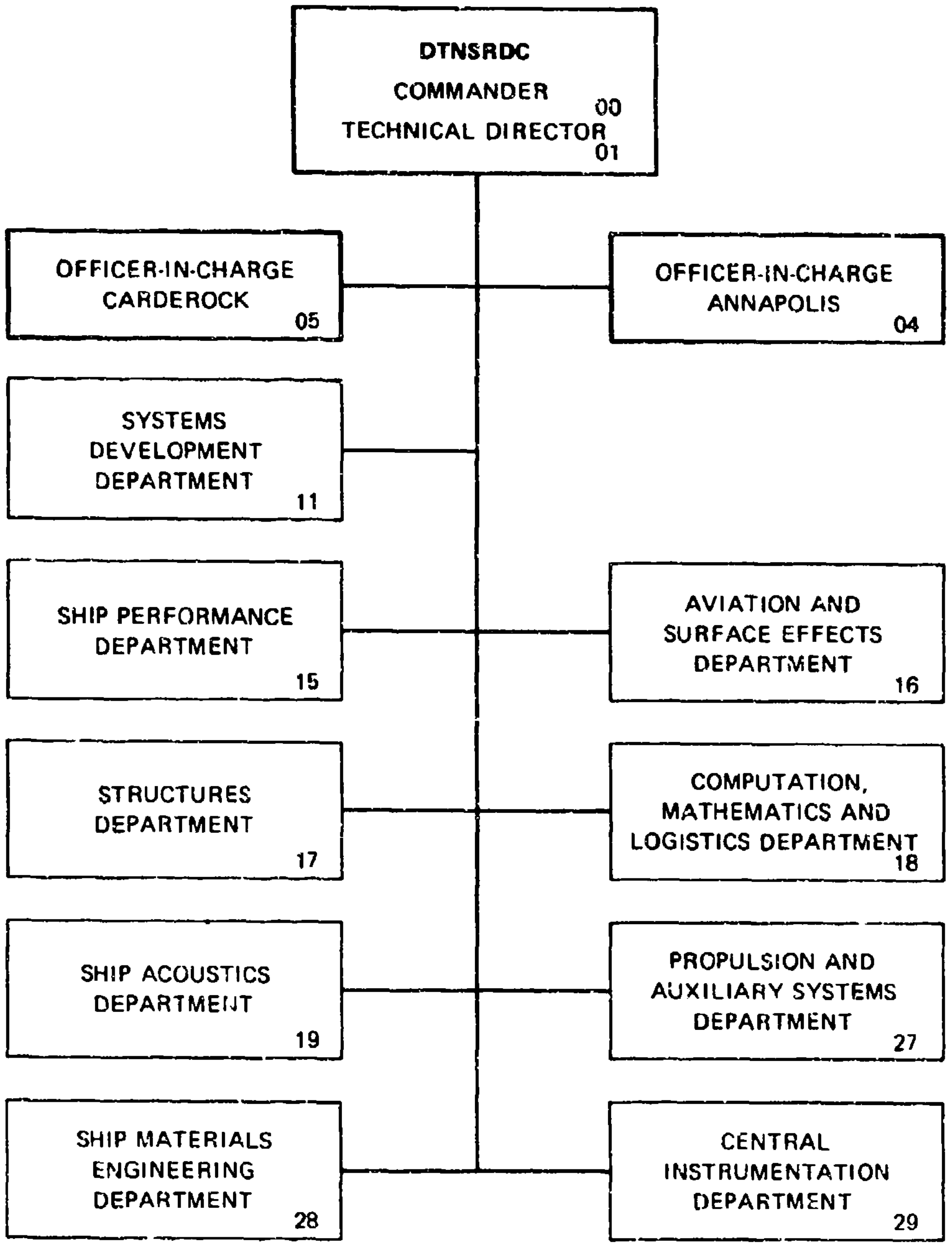
COMPUTATION, MATHEMATICS, AND LOGISTICS DEPARTMENT
DEPARTMENTAL REPORT

February 1985

DTNSRDC/CMLD-85/02

ZOG/VINSON TECHNOLOGY DEMONSTRATION PROJECT
SYSTEM DESCRIPTION, VOLUME I

MAJOR DTNSRDC ORGANIZATIONAL COMPONENTS


DTNSRDC
COMMANDER 00
TECHNICAL DIRECTOR 01
OFFICER-IN-CHARGE CARDEROCK 05
OFFICER-IN-CHARGE ANNAPOLIS 04
SYSTEMS DEVELOPMENT DEPARTMENT 11
SHIP PERFORMANCE DEPARTMENT 15
AVIATION AND SURFACE EFFECTS DEPARTMENT 16
STRUCTURES DEPARTMENT 17
COMPUTATION, MATHEMATICS AND LOGISTICS DEPARTMENT 18
SHIP ACOUSTICS DEPARTMENT 19
PROPULSION AND AUXILIARY SYSTEMS DEPARTMENT 27
SHIP MATERIALS ENGINEERING DEPARTMENT 28
CENTRAL INSTRUMENTATION DEPARTMENT 29

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT APPROVED FOR PUBLIC RELEASE: DISTRIBUTION UNLIMITED
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) DTNSRDC/CMLD-85/02	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION David Taylor Naval Ship R&D Center; 6b OFFICE SYMBOL *(If applicable)* Code 1826	7a NAME OF MONITORING ORGANIZATION
6c ADDRESS *(City, State, and ZIP Code)* Bethesda, MD 20084-5000	7b ADDRESS *(City, State, and ZIP Code)*
8a NAME OF FUNDING/SPONSORING ORGANIZATION Office of Naval Research; 8b OFFICE SYMBOL *(If applicable)* Code 270	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c ADDRESS *(City, State, and ZIP Code)* Arlington, VA 22217	10 SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO 62763N; PROJECT NO RF63521; TASK NO; WORK UNIT ACCESSION NO 11826008

11 TITLE *(Include Security Classification)*
ZOG/VINSON TECHNOLOGY DEMONSTRATION PROJECT: SYSTEM DESCRIPTION, VOLUMES I and II

12 PERSONAL AUTHOR(S)
Donald J. Schmelter, Ron Lupish

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT *(Year, Month, Day)*	15 PAGE COUNT
Final	FROM Mar 81 TO Oct 84	February 1985	300

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18 SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
FIELD	GROUP	SUB-GROUP	

19 ABSTRACT *(Continue on reverse if necessary and identify by block number)*

The ZOG system is a user-modifiable, menu-oriented, rapid response human-computer interface on a network of powerful workstations, the PERQ Systems Corporation's PERQ microcomputers. This document describes how the ZOG system operates. This is not a user's guide but a description of what is behind all of the menu creation and other standard features.

This document is divided into four separate and distinct volumes. It was written this way to best describe the total system while keeping volumes apart so as to make each one accessible without having to go through the others. The first volume is the ZOG System Description. The system description includes a description of:
An overview of the ZOG system, the initialization process, basic system flow, how accessing frames and subnets is accomplished, ZOG utilities, ZOG editors and ZOG agents.
The second volume is ZOG Structures. This volume lists all of the differenct structures used within the code that makes up ZOG.
The third volume is ZOG Files. ZOG Files lists and describes all of the files that ZOG

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a NAME OF RESPONSIBLE INDIVIDUAL Donald J. Schmelter	22b TELEPHONE *(Include Area Code)* (202) 227-1622; 22c OFFICE SYMBOL Code 1826

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

(Block 19) Continued
needs in order to run. These files are in addition to all of the source and executable files that make up the ZOG system.
The fourth volume is ZOG Modules. ZOG Modules goes through ZOG, module by module, describing what is going on.

Each of the four volumes has a different function. The system description is useful for an overall view of how ZOG functions. The structures volume is good for a quick reference of what all of the ZOG records and types are. The files volume is useful to see exactly what files ZOG needs and what they are used for. The modules volume is extremely useful for actually going into the pascal code and seeing how ZOG works on a module level.

ZOG OPERATIONAL DESCRIPTION

Page 11 is intentionally left blank per
Mr. Jack Jeffers, DTNSRDC/Code 1826

ZOG System Description

Table of Contents

List of Figures

ABSTRACT

The ZOG system is a user-modifiable, menu-oriented, rapid response human-computer interface on a network of powerful workstations, the PERQ Systems Corporation's PERQ microcomputers. This document describes how the ZOG system operetes. This is not a user's guide but a description of what is behind all of the menu creation and other standard features.

This document is divided into four separate and distinct volumes. It was written this way to best describe the total system while keeping volumes apart so as to make each one accessable without having to go through the others.The first volume is the ZOG System Description. The system description includes a description of:

an overview of the ZOG system, the initialization process, basic system flow, how accessing frames and subnets is accomplished, Zog utilities, Zog editors and Zog agents.
The second volume is ZOG Structures. This volume lists all of the different structures used within the code that makes up Zog.
The third volume is ZOG Files. ZOG Files lists and describes all of the files that Zog needs in order to run. These files are in addition to all of the source and executable files that make up the Zog system.
The forth volume is ZOG Modules. ZCG Modules goes through Zog, module by module, describing what is going on.

Each of the four volumes has a different function. The system description is useful for an overall view of how Zog functions. The structures volume is good for a quick reference of what all of the Zog records and types are. The files volume is useful to see exactly what files Zog needs and what they are used for. The modules volume is extremely useful for actually going into the pascal code and seeing how Zog works on a module level.

ZOG System Description

This document contains a detailed description of the operation of the ZOG system, Version 29.0, its major software components and their interactions.

1. Scope

1.1. Document outline

This document is organized as follows: There are four seperate and distinct sections. The first is the Zog System Description. The system description includes an overview of the Zog system, a description of the initialization process, Basic system flow, Accessing frames ans and subnets, Zog utilities, the editors and the Agents. Sections two through four are given as appendicies. Section two deals with Zog Structures. It describes the major structures that Zog uses. Section three deals with the files used in Zog. Section four describes Zog module by module.

1.1.1. Notations and Conventions Used:

- The angle brackets (<>) : Generic occurrence of; thus <integer> means substitute any integer: <SubnetName> means any subnet name can be used.
- The square brackets ([]) refer to optional items: [illegible] ,SubnetName] implies that SubnetName may be used, but is not necessary.
- <ModuleName>.<ProcedureName> : refers to proc[illegible]unction <ProcedureName>, which is contained in Modula <ModuleName>.
- Lower case is used for PASCAL variable types; mixed upper and lower is used for ZOG defined types.

- Files are referred to in the same way as for POS utilities: <DeviceName>:<PartitionName>[><SubdirectoryName>]><FileName>
- Debugging calls, and most statistics recording calls are not documented in this manual.
- "*" is a wild card character.

2. Overview

The ZOG system is a specific attempt to implement the concepts of a user-modifiable, menu-oriented, rapid response human-computer interface on a network of powerful workstations, the PERQ Systems Corporation's PERQ microcomputers. Descriptions of this environment as it appears to the user are given in other documents, and will not be repeated here. A functional overview of the system's basic operations is given below:

- Frames and Subnets
- Initialization
- Selection processing
- Action processing
- Net Accessing, local and remote
- Frame Modification: Editing
- Agent Invocation

2.1. Frames and Subnets:

The fundamental unit of structure in ZOG is the frame, which was loosely defined as a screenful of information. However, this definition was really only applicable to ZOG as implemented on a VAX, with standard 24 line by 80 character CRT's as the user interface device. With more sophisticated devices, this definition is clearly inadequate. A perhaps less restricted, but clearer definition of a frame is a collection of closely related textual information structured as a unit inside a window (of arbitrary size).

A subnet, then, is loosely defined as a collection of frames of related content. The frames within a subnet are linked together, typically in a tree-structured manner. However, the linking mechanisms of ZOG are flexible enough to allow for other ways of linking the frames of a subnet together, such as linearly, in a ring, or in a very complex "net" of interconnections. The only "rule of thumb" governing which frames fit in a given subnet is that the frame should bear some informational relationship to the subject matter or function of the entire subnet.

2.2. Initialization

Before ZOG can be run on a given PERQ, certain initialization procedures must be performed. Other than the usual initialization of variables and allocation of memory for the creation of at least the base-line dynamic variables, the most important parts of the initialization process are the following:

- Initializing the PERQ display, and setting up the windows for displaying frames and displaying user messages
- Opening zog.log and exception.log for writing
- Ethernet hardware initialization
- Reading in file subnet.index into a hashed array in memory for later reference
- Reading in auxiliary files, such as top.frame, bottom.frame, help.frame, zog.animate, etc.
- Opening the files containing the top and bottom frames and reading in these frames for displaying

2.3. Selection processing

Selection processing is the primary method by which users traverse ZOG nets. It is also the core of the main ZOG cycle: ZOG basically loops around waiting for user input to process via the selection processor. The act of typing in the selection character of an option or local pad will cause the frame pointed to by that option/pad to be retrieved and displayed in the current window. If the user chooses "next" and there is no "next frame" associated with the selection, then the user is asked if he/she wishes to create one.

The selection processor attempts to match the character the user typed in with the selection character of the options, then the local pads, and finally the global pads. If there is a match, then the next frame field associated with the selection is retrieved. If it is a valid frame, that frame is then displayed.

2.4. Action processing

Actions are the ways in which most things are done in ZOG. Just about anything one can think of doing within the ZOG environment except traversing subnets directly via selection processing, is accomplished, or at least initiated, through an action. A reasonable definition of an action is a low-level operation which typically affects only one frame. Actions are invoked either by typing in a ZOG command at the keyboard, or by having ZOG retrieve and process the associated action string of a frame, local pad, or global pad that the user has selected.

The action processor in ZOG parses the action string, and dispatches to the appropriate procedure to accomplish the operation. Examples of common actions are:

- goto a user-specified frame
- go back one frame in the frame stack
- go to a next/previous frame
- redisplay the current frame
- change to the other window
- create a new subnet, or a new frame
- read/write the zbh form of a subnet

2.5. Net Accessing, local and remote

ZOG deals with structures of frames called (sub)nets. Subnets exist as disk files on various PERQs. In order to do any processing at all, ZOG must be able to read, modify, create and write frames within specified subnets, whether these subnets exist on the user's machine or on any other machine connected via the EtherNet.

The way that subnets of frames are represented within files and the way these are accessed is described below:

2.5.1. File representation of subnets

A subnet on the PERQ is simply a disk file. This file is structured to facilitate the accessing of any given frame in the subnet. Each frame in the subnet is allocated 10 disk blocks within the file. Thus, to access frame 27 of a particular subnet, ZOG does 10 block reads of this file, starting at block 270 of the file. In order to conserve disk space, however, unused sections of a frame's 10 block allocation are "free" to the disk's file system; thus, the subnet file has "holes" in it.

These subnet files are named <SubnetName>.pri, the "pri" being an abbreviation for "primary." Because of the fact that these files have "holes" in them, they are not directly transportable for installation on other machines, or for backup purposes; the usual file utilities of copy, edit, or move to floppy disk do not handle these files correctly. Thus, it is necessary to transform each .pri file into a form which is transportable, that is, without the holes. Utility programs have been written to transform these files into a transportable form. The transportable version of the subnet files are named <SubnetName>.zbh.

2.5.2. Individual frame representation in .pri/.zbh file.

An individual frame is written out to disk in a modified "BH" format, called ZBH format. This format was developed at CMU as a way of storing variable types of records in an ASCII disk file. Each item in a logical record is stored as a line of the form:

```
+<char>+ [<ASCII string>]<EOL>
```

where <char> is a single ASCII character which encodes the type of data stored on that "line", the optional <ASCII string> can be text, numbers, codes or special characters, and <EOL> is the end-of-line character(s).

2.5.2.1 ZBH Codes for Frame Header

These are coded in BaseLib.ParseFH

These codes are for the non-text information contained in the frame. Other than the protection code, this information is maintained automatically by ZOG.

ZBH Item	Representation of string in file
+A+	Frame Id String
+B+	Created by Agent Boolean
+b+	Modified by Agent Boolean
+c+	Creation Date Integer String
+M+	Modifier Name String
+m+	Modification Date Integer String
+p+	Protection Code Character
+t+	Modification Time Integer String
+U+	(List of) Frame Owners String
+V+	Frame Version Number Integer String
+Y+	(List of) Frame Accessors String (not used)
+Z+	End of Frame Header Marker

2.5.2.2 ZBH Codes for Frame Body

The code for this is in NetHandl.ParseF. The frame body consists mostly of text fields. This information is stored in Frame Title, Frame Text, Options, Local Pads Order. Within each Item of the frame, the order is Item Marker & Selection Character, Item Text, Item Position, Next Frame, Action.

ZBH Item Representation of string in file

+C+	Frame/Selection Comment String
+E+	Frame/Selection Expansion Area String
+F+	Selection's Next Frame String
+G+	Global Pads Frame String
+I+	Frame Text Marker String (Normally Empty)
+L+	Local Pad Marker & Selection Character Character
+O+	Option Marker & Selection Character Character
+P+	Item's Position Pair of Integer Strings
+T+	Frame/Selection Text String
+X+	Frame/Selection Action String
+Z+	End of Frame Body Marker
+<other character>+	Extra Fields String

2.5.3. Local frame access

Frames physically on a user's machine are accessed via routines in the NetServer software. Once ZOG determines that a frame is in fact on the user's machine (through a lookup in the internal representation of subnet.index), a ReadFrame process is initiated. The frame header (the first block of the frame's 10 block allocation) is read in and parsed into the frame header structure, then the remaining blocks (the frame body) are read in and the frame's internal structure is filled out.

After a frame has been created or modified, the process for writing it out is similar; the frame's header information is retrieved from the record, and is written out a "line" at a time into relative block 0, then the same happens for the rest of the frame structure, into relative blocks 1-9.

2.5.4. Remote frame access

The process of reading frames which exist on another machine on the ethernet network consists of several steps. The requesting machine sends out a message to the machine which has the needed frame. If the machine with the frame is running and has received the message, it sends an acknowledgement to the requesting machine. In addition, the remote machine starts its own disk I/O to read a frame, just as for local frame access. The whole process on the remote machine is initiated via an "interrupt" or exception handling routine; its main ZOG processing is interrupted.

When the frame read has been completed, the frame information is put into ethernet compatible packets and sent to the requesting machine, with all the appropriate protocols. The requesting machine then parses these ethernet packets into its internal frame structure and sends it to the higher level routines just as if it had been read locally. Meanwhile, the remote machine returns control to its own primary ZOG process.

Writing of frames on a remote machine is handled similarly.

2.6. Frame Modification: Editing

Even though the ZOG editors are invoked through actions (see above) they will be discussed separately here, since they figure prominently in the overall use of the ZOG system.

2.6.1. The ZOG Editor, ZED

ZED allows ZOG users to modify frames. Commands exist within ZED which provide for the usual editing functions of insertion, deletion and change/replacement. In addition, ZED has commands for accessing a frame's *invisible* areas, such as next frame, frame/selection comments, actions or expansion areas. Other commands allow for formating and justifying textual information in a frame. Specialized commands provide simple ways to create and remove selections within a frame.

ZED creates its own local copy in memory of the frame being edited, in a frame structure pointed to by pointer variable EdFP. Only when ZED is exited for updating is this modified frame written out to disk.

2.6.2. The Slot Editor, SLED

Many of the applications in ZOG are contained in agents (see below). These agents, however, typically require one or more user-specified parameters, such as subnet name, output file name, way of traversing subnets, and the like. Special frames, called environment frames, have been created to provide a simple, structured mechanism for users to specify this information. These environment frames use their options as *slots* in which the user is to type in information.

SLED, then, is the editor which, in conjunction with the structures *behind* the environment frame, provides the means for users to fill in these slots. These structures are "hidden" behind these slots in "slot frames", accessed through the alternate next frame mechanism, via the selection's expansion area. Mechanisms exist for toggling "yes/no" slots, for selecting from a menu of pre-determined allowable responses, and for verifying subnet names, dates, and the like. Although ZED can be used for this purpose as well, it provides no checks or aids to the user. Thus, even experienced ZOG users will tend to use SLED for filling in environment frames rather than use ZED.

SLED, like ZED, maintains its own private copy of the frame being edited in a Frame record pointed to by SLedFP.

2.7. Agent Invocation

Agents are basically processes within ZOG which *know* about ZOG structures. Typically agents operate on subnets of frames, or portions thereof. Agents are used to accomplish a variety of tasks within ZOG. Parameters to agents are given via the environment frames filled in by the user. The agents are invoked via a special action which has as its parameters the agent name being invoked, the top frame of the agent's subnet, and the frameid of the environment frame for the agent.

2.7.1. Utility agents

This is the category with by far the largest number of agents. Some of these are user-oriented utilities, others are more sophisticated utilities meant primarily for ZOG experts.

2.7.1.1 User utilities

These are the utilities that everyday ZOG users are likely to use. They accomplish the following tasks:

- Copy trees of frames
- Find frames matching certain criteria, such as creator, creation date, or containing a user-specified string
- Create an index to a user's (subset of) subnets
- Create an "old" copy of a frame for backup purposes
- Check (and repair if necessary) parent and top local pads in a tree of frames
- Create a new subnet
- Send Mail to another user
- Do a global string replacement in a subnet

2.7.1.2 Backup utilities

These agents are for system maintainers to create a backup copy of subnets on streamer tape or floppy disk. There are several ways of doing so:

- Archiving subnets on system-formatted floppies; local machine only
- Writing out subnets in their transportable (zbh) form, suitable for backing up on floppy disk or streamer tape
- Performing a writing out of subnets from a remote machine into their ZBH form

2.7.1.3 ZOG System utilities for experts and system maintainers

These agents allow *sophisticated* ZOG users to perform the following tasks:

- Setting up a batch stream to invoke a series of agents
- Insuring that secondary copies of subnets are fully updated
- Creating lists of subnets, alphabetically or by machine name
- Obtaining the highest frame number in a subnet
- Changing the owner and/or protection of frames in a subnet
- Modify the links of a frame to a new frame, with new parents

2.7.2. Task management agents

These agents are essentially for managing *task trees*, that is, trees of task frames, the leaf frames of which describe specific tasks to be performed by specified accomplishers in a specified time period. Modifications to these frames are made with the ZOG Editor, ZED. These agents allow managers to:

- Initialize a new task tree
- Generate a hard-copy or frame (or a ship-specific) form of a plan
- Create a specific task tree from a generic one or vice versa
- Update the upper levels of a task tree to reflect changes in the leaf frames (specific tasks)
- Modify Starting and Ending Times and Dates in a specific task tree to reflect overall scheduling changes
- Create copies in various formats of (sections of) the SORM

2.7.3. Text file outputting agents

Often it is necessary to obtain a text file version of a subnet or portion thereof, perhaps for printing a hard copy. Agents exist which will allow users to:

- Create a readable sequential text file document from a tree of frames, including indentation, table of contents, pagination, etc.
- Create a *picture* of a frame in text file form, which is useful for documenting items in ZOG
- Create a compilable PASCAL source file from a tree of PASCAL code frames created within the ZOG Programming environment

Page 11 is intentionally left blank per
Mr. Jack Jeffers, DTNSRDC/Code 1826

3. Initialization of ZOG

After being loading by the PERQ Loader, ZOG first sets its main segment to be UnSwappable to prevent any of the main ZOG code from being swapped out, as would happen during an Ethernet interrupt. This was found to be necessary during ZOG's development, when segment faults were experienced.

ZOG next calls ZInitExit.InitZOG to perform all the remaining initializations necessary for ZOG to run. InitZOG in turn calls procedures from each of the main modules of ZOG which perform specific initializations for their own particular subsystems.

Note: It has been found that the order in which the initialization routines are called is very critical. Programmers are well advised to be very careful when changing the orderin which these initialization procedures are called.

When the initialization is completed, ZOG simply enters its main loop, getting user's selections and processing them.

3.1. ZInitExit.InitZOG

InitZOG is essentially a dispatcher of all the various initialization routines. It also manipulates the globals InInitialization and InZOGLogin to communicate to *the world* that initialization is in progress. It calls the following routines in the order shown:

- ZOGVersion.InitZOGVersion
- ZLogFile.InitLogFile
- ZCanvas.InitZCanvas
- ZTrace.InitTrace
- ZParse.InitParse
- ZDisplay.SetTerm
- ZUser.InitUser
- ZDisplay.InitDisp
- ZInitOthers.InitOthers
- ZWind.InitWind

- ZVideo.InitV
- ZOGNetServer.InitZOGNetServer
- NetHandl.IniNetHandl
- ZParse.GetRemArg
- ZEint.EIntReady
- ZLogFile.LogInit
- ZInitExit.SetUpFrames

3.1.1. ZOGVersion.InitZOGVersion

This procedure stores the current ZOG Version number into the global variable ZOGVersNumber. This procedure is called only once in all of ZOG. Several other routines, however, access ZOGVersNumber, specifically, routines in ZLogFile (to label the log files with the current ZOG Version number), ZDump (similarly for the exception logging routines in the log files and in the exception.log file), ZStats (for labeling statistics frames logged during ZOG sessions), ZDaction (to label the *who* command (↑dz)) and ZCanvUtils (for displaying t he current version number at the top of the active window).

3.1.2. ZLogFile.InitLogFile

This procedure does a rename of the log files, zog.log<n>, to zog.log<n + 1>, where n goes from nothing to 2. These files are located in partition :zog, subdirectory log. If any of these renames fail, the failures are caught by a local exception handler, and control is returned back to InitLogFile.

File :zog>log>zog.log is then opened for writing as the current logging file.

Finally, the local boolean LogFileSig is set to true to allow ongoing event logging and the local boolean ZOGInited is set to false, so that only initialization is logged. ZOGInited is later set to true in ZLogFile.LogInit, which logs the fact that ZOG initialization has been completed.

3.1.3. ZCanvas.InitZCanvas

This routine initializes the interface between ZOG and the PERQ Screen module, as well as all of the lowest level I/O utilities as follows:

- Screen.ScreenReset to initialize the Screen module and clear screen
- ZCanvas.SetCanvFunc to set black/white background color

- ZCanvUtils.SetZOGCanvas to create windows on the screen for ZOG use
- ZCanvas.SetChFunc to set text display function to Replace mode
- ZCanvUtils.IniCanvPtr to read in the ZOG mouse pointer images
- ZCanvas.SetCanvPtr to set pointer image to solid arrow
- ZCanvas.SetPtrCh to place pointer at the top left of the current window
- Initialize mouse-related variables
- ZPoll.InitPoll to initialize input polling facility
- Set EXPORTed boolean, FrameCanvas, to true to indicate that a frame is being displayed
- Set EXPORTed variable, UsrDispLine, to 23, the default line in a frame window for displaying user messages
- ZCanvas.SetAltCursor to initialize the alternate cursor function to the box around the options mode

3.1.4. ZTrace.InitTrace

Most of the ZOG code has built-in tracing commands which will print out debugging statements in the user display area conditionally depending on the value of a tracing boolean. There are currently 31 of these tracing booleans, most of which begin with the letters "Trc". These variables are defined in and exported from module ZTrace.

InitTrace, then, merely sets all 31 of these global booleans to false; thus, the default condition for ZOG is "No Tracing". One or more of these tracing booleans can be turned on by specifying the proper letter(s) as a switch to the "zog" command to the shell. In addition, the ZOG user can toggle these booleans with the Control-At<char> action. See ZAAction.CntlASel and ZParse.ProSwitches.

3.1.4.1 List of Tracing Booleans:

A list of the variable names, along with the code letter that toggles them via the ↑at action. See ZParse.ProSwitches.

- MTrace 'm'
- PTrace 'P'
- TimeSig 'T'
- TrcBaseLib 'B'

- TrcEtherReq 'Q'
- TrcFld 'f'
- TrcLevel 'l'
- TrcNetHandl 'H'
- TrcNetInsert 'I'
- TrcNetLib 'N'
- TrcMakeDel 'M'
- TrcNetOption 'O'
- TrcNtStack 'C'
- TrcNetString 'r'
- TrcSubnet 'S'
- TrcZAction 'a'
- TrcZAgent 'g'
- TrcZBack 'b'
- TrcZBHIO 'h'
- TrcZCanvas 'c'
- TrcZDisplay 'd'
- TrcZEdit 'e'
- TrcZIO 'i'
- TrcZOGNetServer 'Z'
- TrcZParse 'p'
- TrcZSel 's'
- TrcZTrace 't'
- TrcZUser 'u'
- TrcZVideo 'v'

- TrcZWind 'w'
- TrcZOG 'z'

3.1.5. ZParse.InitParse

The purpose of this routine is to obtain any switches added to the user's "zog" command line to the shell. However, this is only done if ZOG itself is NOT the currently active shell of the PERQ. In many situations, it is desirable to make ZOG the shell, so that if ZOG crashes, it will come up again automatically through the PERQ's own shell loading routines. ZOG is easily made to be the shell by adding a line to the user's profile file after the login command of */shell = :boot>zog*.

Examples of switches the user might add are tracing switches to turn tracing on for a particular set of routines, or to specify an external terminal type.

This routine examines the POS global variable, ShellName, to see if ZOG is the current shell. If not, procedure ZParse.GetArg is called to retrieve any user supplied arguments to zog.

3.1.6. ZDisplay.SetTerm

ZOG has the capability of sending every character displayed in a window to an external video display terminal. This facility is mostly used to allow for projecting the image of ZOG frames on a large screen via the ElectroHome device (using the Concept 100 terminal) or a projection television (using the video output of a VT-100 type of terminal).

SetTerm sets the global variable, TermK, which indicates the auxilliary terminal type, to a desired value. This variable tells the whole system what type of video terminal is connected to the PERQ's RS-232 port. In general, its default value will be 'null', indicating that there is no external terminal. If there is an external VDT, specified by the user, then this routine will initialize it to 9600 baud, and send it the codes to clear its screen.

Valid values for TermK are (currently):

null	No terminal
'c'	CONCEPT 100
'v'	DEC VT100 and compatible family of VDTs.

3.1.7. ZUser.InitUser

Initialize the internal variables needed to run the ZOG User Display windows. This routine will:

- Allocate storage for the user display text buffer.
- Call ZUser.InitUsrDsp:
- Set Exported variable UserSig to false, indicating that the full-screen User Display is not visible.

3.1.7.1 Allocate storage for the user display text buffer.

3.1.7.2 ZUser.InitUsrDsp:

- Initialize the pointers into the text lines buffer.
- Set the EXPORTed variable PromptLength to 0, indicating that there is no prompt to be displayed in the user display line.

3.1.7.3 Set Exported variable UserSig to false, indicating that the full-screen User Display is not visible.

3.1.8. ZDisplay.InitDisp

This routine merely initializes the exported string variable, BlankLine, to 80 blank characters. This variable is used throughout ZOG, especially in the editors. Having it initialized once here and exported saves having separate blank line strings scattered throughout the system.

3.1.9. ZInitOthers.InitOthers

Module ZInitOthers had to be written as a way of invoking many additional initialization routines via ZInitExit.InitZOG. This was necessary due to the limitations on the number of modules allowed to be IMPORTed by a given module imposed by PERQ PASCAL; ZInitExit was already near that limit. ZInitOthers invokes the following procedures to continue the process of ZOG initialization:

- ZBack.InitBack
- BaseLib.IniBaseLib
- NetMakeDel.IniNetMakeDel
- NetStack.IniNetStack
- ZAction.InitAction
- ZAgent.InitAgent
- ZEdit.InitEdit
- ZSled.InitSled

- IncDisp.InitIncDisp
- ZPoll.InitPoll
- AirStat.initAirStat
- ApLover.InitMailPsc
- { NetPerqCodes.InitNPCodes }

3.1.9.1 ZBack.InitBack

This module is responsible for maintaining the backup stack of frames visited. The initialization routine does only two things:

- Initializes the module's own private backup list pointer variable, BackSavP, to NIL
- Initializes the EXPORTed boolean, Return, to false, indicating that ZOG is NOT inside the "return" function (global pad "ret")

3.1.9.2 BaseLib.IniBaseLib

BaseLib is the home of most of the general purpose utility procedures and functions in ZOG. Among other things, BaseLib provides procedures for maintaining ZOG's own private lists of available memory for dynamically allocating space for certain ZOG structures, specifically, a list of unused *string15* records and a list of unused *Frame Header* records.

The initialization procedure, IniBaseLib simply initializes some local variables, and allocates memory for a buffer.

- Initialize BaseLib's local string15 pointer variable, Fs15PSav to NIL
- Initialize BaseLib's local frame header pointer variable, FHPSav to NIL
- Allocate a one-page buffer for possible use by procedure AppStrFile
- Set the exported default protection variable, DefProt to -1, indicating protection to be used in a copied frame

3.1.9.3 NetMakeDel.IniNetMakeDel

Module NetMakeDel is the interface to the NetServer for creating and deleting frames. IniNetMakeDel initializes some of the variables used in this process.

- Initialize the local pointer variables, FsPSav, SelPSav, and FPSav, to NIL. These variables point to lists of allocated but unused string records, selection records and frame records. The Rel(ease) and Cr(eate) procedures in this module maintain these lists locally to avoid having to do many PASCAL new and dispose commands.

- The EXPORTed string variable, DefGPad, which stores the Frame ID of the default Global Pad frame for new frame creation, is initialized to *GPads1*, the current ZOG default global pad set...

- Memory.CreateSegment is called to create a separate segment (NetSeg) for this module's own private use to avoid using the default data heap.

- Memory.SetMobility is called to make segment NetSeg UnMoveable. Both Memory calls are necessary to avoid segment fault problems while processing remote frame requests.

3.1.9.4 NetStack.IniNetStack

The NetStack module provides programmers with a set of procedures for maintaining a sa[illegible] stack of frames being processed. Although this module will eventually be replaced by the more powerful and efficient routines contained in module StackLib, it is still imported by a number of modules, especially the older modules for SORM maintenance.

IniNetStack initializes this stack for later use with the default global frame pointer variable FPX by calling procedure InitFStk. This routine does the following with the EXPORTed stack record variable FStkX:

- Set the top FrameId in FStkX to null

- Set the pointer to the top of the stored frameid list to NIL

- Set the Frame record pointer in FStkX to be the global pointer variable FPX, EXPORTed from Module NetHandl. This pointer variable is (was) used throughout ZOG, especially in agents.

3.1.9.5 ZAction.InitAction

The purpose of this procedure is to initialize the Action Processing part of ZOG. Currently, this involves two steps:

ZDaction.InitDAct

ZVideo.InitV (For details see p. 24)

3.1.9.5.1 ZDAction.InitDAct

This procedure initializes some data structures used locally within Module ZDAction only. These variables names all start with "Act".

These initializations are:

- The pointer to action strings being executed, ActFsP, is set to NIL

- A local frame record is created via a call to NetMakeDel.CrFP and pointed to by ActFP

- A second local frame record, used during the frame deletion process, is created and pointed to by ActFP2

- A local frame header record is created by a call to BaseLib.CrFHP and pointed to by ActFHP

3.1.9.5.2 ZVideo.InitV -Initialize the Video Disk

Note to programmers: this procedure is called again later directly from ZInitExit.InitZOG Thus, it should probably eventually be removed from here.

3.1.9.6 ZAgent.InitAgent

Currently this routine is a dummy procedure, and does nothing, since any agent invoked will perform its own local initialization. The procedure was left in to allow for any general initialization for agents which may be done at a later time.

An initialization of the StackLib data structures could be inserted here, although, as for agents, these structures are currently set up to be initialized each time the routines are going to be used. Presumably a global structure could be created, initialized once, and used as needed by agents and other ZOG routines.

3.1.9.7 ZEdit.InitEdit

InitEdit is called to initialize the data structures which will be used throughout the ZOG Editor, ZED. It does the following initializations:

- Create an edit frame record for storing a copy of the frame being edited, via a call to NetMakeDel.CrFP

- Allocate memory for a ZED-wide global character buffer pointed to by BufP (defined and declared in ZEDDefs); Set the buffer's initial length to zero (empty buffer)

- Set the ZED-wide global boolean, NetWalking, to false; (This boolean, EXPORTed from ZEDDefs, means the user has temporarily left the editor, and may want to resume editing in the frame last edited.)

- Set the ZED-wide global boolean, XCHG, EXPORTed from ZEDDefs, to false

- Set the ZED-wide global variable, OldWindow, EXPORTed from ZEDDefs, to 1

- Set the default length (in lines) of a frame to be edited

3.1.9.8 ZSled.InitSled

Although the Slot Editor, SLED, was initially created for editing Agent Environment frames, it also proved useful for editing AirPlan input frames within the AirPlan expert system environment, since it constrains the types of entries allowed in given *slots* in a frame. Thus, SLED initialization also includes initializing some AirPlan support data structures.

- Create a record for frames being edited via NetMakeDel.CrFP; this newly allocated record is pointed to by SledFP
- Set the local AirPlan boolean, AirPlanUp to false, indicating that AirPlan is not currently running
- Allocate memory for an AirPlan message data structure, pointed to by ZOGMsgP; This structure is defined in Module ZOGMsg
- Create a local pointer for AirPlan input messages as a recast of ZOGMsgP into an AirInPTyp variable, AirInP. AirInPTyp is defined in Module ZOGMsg.

3.1.9.9 IncDisp.InitIncDisp

The Incremental Display module provides for automatically updating a displayed frame on a user-specified schedule. This has proved very useful for the AirPlan system, where output frames are updated quite often from the AirPlan expert system machine. As other users are observing these frames, they are automatically redisplayed periodically with changes highlighted in reverse video.

This module is initialized by InitIncDisp as follows:

- Create a scratch Frame Header record, pointed to by FHP, via a call to BaseLib.CrFHP
- Set IncDispSig to false, indicating no incremental display currently on
- Set StopReading to false, indicating that the *source* machine is up, so that ongoing reading of the displayed frame can continue
- Initialize the bad read count, BadReads to zero
- Set IncDispTime to 10, the default value

3.1.9.10 ZPoll.InitPoll

ZOG has a built-in polling function, which is implemented through ZCanvas.ZPollCanvas. This routine will invoke the specific poll function every PollInterval jiffies (1/60 second). This is initialized in IniPollProc, below.

There are three separate polling functions which can currently be installed in ZOG. The specific polling function desired is installed via the link stream for ZOG by overtly specifying the polling module to be linked into the ZOG RUN file.

The three polling initialization functions (and their modules) are listed here:

- ZPollSnap.IniPollProc : No longer used
- ZPollAir.IniPollProc : Initialize Polling for AirPlan message handling
- ZPollProc.IniPollProc : Generic Polling initialization for no polling function at all

3.1.9.10.1 ZPollSnap.IniPollProc : No longer used

This procedure originally initialized the Statistics *snapshot* function, to record user and ZOG activity over the past polling interval. Statistics snapshotsd were disabled at before this version was released.

.9.10.2 ZPollAir.IniPollProc

This procedure initializes the AirPlan interface polling function, to dump any queued output from AirPlan to the AirPlan Ops PERQ. It does this as follows:

3.1.9.10.2.1 Turn off debug tracing of polling functions

3.1.9.10.2.2 Set the Polling interval variable, PollInterval, to be equal to 600 *jiffies* corresponding to 10 seconds. Thus, an AirPlan message may be sent every 10 seconds, if there is one (or more) queued up.

3.1.9.10.2.3 AirOutput.InitAirOutput (Initialize the AirOutPut Module)

The full i[illegible]tialization of the AirPlan outputting function occurs here, as follows:

3.1.9.[illegible].3.1 Call AirLib.InitAirLib to initialize the AirPlan Library Module

This is the procedure which initilializes the Library of AirPlan procedures.

- Create a separate 45 block segment for AirPlan packets, using Memory.CreateSegment;
- Make this segment unswappable, using Memory.SetMobility;
- Set the local pointer to the list of saved AirPlan messages, SavAirP, to NIL;
- Allocate memory for the maximum allowed number of records for saved AirPlan message packets within the local data segment, and add each one to the linked list of saved message records using procedure AirLib.ErPack.

3.1.9.10.2.3.2 Initialize all the EXPORTed AirPlan Message Packet Pointers to NIL; These Packet types (AirPackTyp) are defined in AirPlan.defs.

3.1.9.10.2.3.3 Initialize the MessageWaiting boolean, EXPORTed from Module AirCom, to false, indicating no messages waiting to send;

3.1.9.10.2.3.4 Initialize the Statistics recording variable, OutCycleNum, EXPORTed from Module AirCom, to 0;

3.1.9.10.2.3.5 Initialize the global event counter variables, EXPORTed from Module AirCom;

3.1.9.10.2.3.6 Set the AirPlan tracing boolean, TrcAirOutput, EXPORTed from Module AirCom, to false.

3.1.9.10.2.4 Turn polling on via ZPoll.PollOn

3.1.9.10.3 ZPollProc.IniPollProc

ZPollProc is simply the *generic* polling procedure to be linked into ZOG when no polled functions are to be done. Thus, the only thing that ZPollProc.IniPollProc does, naturally, is to turn all polling off, via ZPoll.PollOff.

3.1.9.11 AirStat.InitAirStat

The initialization procedure for the AirStat module merely initializes all the event counts to 0; it also initializes the Launch and Recovery sequence strings to a single blank.

3.1.9.12 ApLover.InitMailPac

This procedure allocates an initial amount of memory for the AirPlan message (or "Mail") packets records. These structures are what are used to send AirPlan *updates* to the Ops7 PERQ running the AirPlan expert system.

InitMailPac does the following:

- Set the local AirPlan boolean, AirPlanUp to false, indicating that AirPlan is not currently running
- Allocate memory for an AirPlan message data structure, pointed to by ZOGMsgP; this structure is defined in Module ZOGMsg
- Create a local pointer for AirPlan input messages as a recast of ZOGMsgP into an AirInPTyp variable, AirInP. AirInPTyp is also defined in Module ZOGMsg.

3.1.9.13 { NetPerqCodes.InitNPCodes

}This module is not used in current versions of ZOG, even though it is imported into ZInitOthers. Its original purpose was to create a table of numerical codes versus PERQ Machine name. This scheme however, has been replaced by a more efficient means, as now implemented in the NetServer code.

3.1.10. ZWind.InitWind

Create and initialize the records containing the data linking ZOG frames to their display windows on the screen. These records include:

- Frame pointers for the Frame and the Global Pads
- Frame IDs for the Current Frame, Global Pads, and Name of displayed frame
- Backup stack pointers
- Selection character used to get to this frame

3.1.11. ZVideo.InitV

The PERQ has an extra RS232 port which can be accessed via procedures provided by the PERQ Operating System. Module ZVideo provides a software interface between ZOG and a videodisc player connected to this port. This allows users to directly, or via specialized actions, have the videodisc player display specified stills or clips. The videodisc feature is useful in creating and using detailed operations, maintenance and repair manuals, such as those now used for the Aircraft and Weapons Elevator systems on board the USS CARL VINSON.

3.1.11.1 First check to be sure that a video terminal is not intended to be used; if so, then exit, since that indicates that the videodisc player is not connected to the RS232 interface

3.1.11.2 Use PERQ system utility RS232Baud.SetBaud to set the baud rate of the PERQ's RS232 port to 1200 baud

3.1.11.3 UEI.VideoInit

The UEI (Universal External Interface) module provides low-level support for accessing the PERQ's RS232 port. Procedure VideoInit assumes that there is a VideoDisk player connected to this port. It resets and activates the port, then, using the Video Disk procedures within the module, it initializes the VideoDisk unit itself. These procedures are called as follows:

- UEIReset
- UEIActivate

- VDWriteProgram
- VDPlay
- VDStop
- VDClr
- VDEndProgram
- VDRunProgram

3.1.12. ZOGNetServer.InitZOGNetServer

This procedure is the gateway to initializing the entire set of modules which initiate local and remote subnet access and service remote requests via the PERQ Ethernet hardware. In addition to initializing the necessary data structures, the EtherNet hardware itself is reset and made ready to send and receive packets of messages.

This procedure also builds the database of available remote machines, and the index of subnets by remote machine.

3.1.12.1 Details of the operation of ZOGNetServer.InitZOGNetServer

- Initialize EXPORTed variables, MaxZOGPort and array LoggedIn[i];
- Read FrameId of Top Frame into EXPORTed variable, TopOfNet.
- ZAccessProcs.InitZAccessProcs { Initialize Module ZAccessProcs }
- NetServ.IniServ { Initialize Module NetServ }
- ZNet.InitZNet { Initialize Module ZNet }
- Allocate memory for local buffers and messages lists;
- ZOGMsg.InitZOGMsg { Initialize Module ZOGMsg }
- Load up EXPORTed variables identifying current machine and user;
- ZOGNetServer.BuildServers { Build Net Servers Database }
- ZOGNetServer.BuildSubnets { Build Subnet Database }

3.1.12.1.1 Initialize EXPORTed variables, MaxZOGPort and array LoggedIn[i];

3.1.12.1.2 Read FrameId of Top Frame into EXPORTed variable, TopOfNet.

3.1.12.1.3 ZAccessProcs.InitZAccessProcs

The only thing this routine does is to create a frame header record pointer, local variable FHPSecUpd, for use in checking the version number of a frame when a secondary update fails. This variable is used in procedure ZAccessProcs.CloseFrame.

3.1.12.1.4 NetServ.IniServ

IniServ initializes a number of variables local to Module NetServ:

- Initialize all 128 entries of local Subnet Hash Table, SnTable, to NILs;
- Initialize the local pointer to top of open frame record list, OpnTopP, to NIL;
- Initialize the local pointer to open frame record list, OpnSavP, to NIL;
- Initialize the local pointer to saved frame header record list, FHSavP, to NIL;
- Initialize the local pointer to saved owner entries list, OwnsavP, to NIL;
- Create temporary local frame header record pointed to by FHPTmp;
- Allocate memory for temporary frame header buffer and output buffer page, local pointer variables FHBPTmp and ZeroBufP, respectively;
- Initialize ZeroBufP to all (long word) zeroes.

3.1.12.1.5 ZNet.InitZNet

This routine merely allocates memory for two local pointers to copies of ethernet messages:

- Allocate memory for outgoing ZOG messages pointed to by OutMsgP;
- Allocate memory for incoming ZOG messages pointed to by InMsgP.

3.1.12.1.6 Allocate memory for local buffers and messages lists;

This is done using the PASCAL "new" intrinsic foi the following records:

- NetFHBP: { Frame header buffer for local use }
- NetFBP: { Frame (body) buffer for local use }
- InMsgP: { Input message list for remote EtherNet requests }
- OutMsgP: { Output message list for replying to remote requests }

3.1.12.1.7 ZOGMsg.InitZOGMsg

Module ZOGMsg contains the software for handling the EtherNet message traffic. Communications on the EtherNet can be asychronous, with no response expected or sychronous, with some message expected in response to the message sent. InitZOGMsg initializes the data structures and allocates memory necessary for these communications, returning true if there were no problems. Specifically, it does the following:

3.1.12.1.7.1 Initialize EXPORTed boolean Debug to false, indicating no debug messages;

3.1.12.1.7.2 Make its own current segment UnSwappable via Memory.SetMobility;

3.1.12.1.7.3 Get name of local machine;

The first line of file EFileName (:boot>EtherNet.Names, q.v.) is read into variable LongName. If necessary, this name is truncated to have a length of 15 characters, then it is moved to private variable, OurName, for use later by function GetMyName. Finally, it is moved to private variable, OurNameUpper, which is then converted to all upper case, for use later in PERQ name comparisons.

3.1.12.1.7.4 Create its own separate data segment, EtherSeg, for message records;

3.1.12.1.7.5 Allocate memory for variables within segment;

3.1.12.1.7.6 Allocate memory for structures to receive message packets in;

3.1.12.1.7.7 Allocate memory for EXPORTed variable, OurAddrPtr;

3.1.12.1.7.8 Make data segment EtherSeg UnSwappable and UnMoveable;

3.1.12.1.7.9 Reset internal EtherNet table (IMPORTed from IO - Others) interrupt mask to Enabled (i.e. to be = E10IntMask (= 4096));

3.1.12.1.7.10 Get EtherNet addre_s of local machine;

Routine Ether10IO.E10GetAdr is called to return the EtherNet address of the local machine. (This 48 bit address is hard-wired into the machine's EtherNet board.) The address is stored in three word-size integers in private record OurAddress. The bytes of each of these integers need to be swapped to form the proper address from that returned by E10GetAdr.

In addition, local record pointer variable, OurAddrPtr, defined to be a pRecEtherAdd type (see Ether10IO Module) gets its LowAddress item set to the low order word of OurAddress, and its MultiCast Command Byte, MCB, set to MltCstNone (= #377), indicating no multicast packets are to be received.

3.1.12.1.7.11 Reset the EtherNet, and repost any pending messages in circular list, CurERecP.

3.1.12.1.8 Load up EXPORTed variables

The following EXPORTed variables (globals to the rest of ZOG) are initialized:

- CurAddr is initialized to the EtherNet address of the *current* PERQ, that is the local PERQ or the PERQ doing the initialization itself, via a call to function ZOGMsg.GetMyAddr;
- CurName is initialized to the local PERQ's name (as contained in file :boot>EtherNet.Names), via a call to function ZOGMsg.GetMyName;
- CurNameUpper is initialized to CurName, converted to upper case (used for name matching);
- CurUser is initialized to CurUserName, IMPORTed from POS Module, System (this gives the name of the currently logged in user);
- CurUserPort is initialized to CurAddr, above.

3.1.12.1.9 ZOGNetServer.BuildServers

This routine is used to build an internal table of EtherNet Servers (i.e., remote PERQs or Nodes) in the distributed ZOG network. The file AllZOGServers (:zognet>Net.Servers) is read in one line at a time. For each line in the file, a record in the EXPORTed array of Server status, Servers, is created. Each of these records contains the name of a PERQ (i.e., a Node on the EtherNet network), its EtherNet address and a boolean indicating whether or not the given PERQ is up and running. The entries for all machines in this list, except the current one (i.e., the PERQ doing this intialization), are initialized with an EtherNet address of 0, and as being down. Only when an EtherNet probe of that machine is done is this record updated otherwise.

Currently, distributed ZOG can support a maximum of 256 PERQs acting as EtherNet servers.

In addition, this procedure also generates a count of the available EtherNet Servers. This count is stored in the EXPORTed variable, CurMaxServer.

3.1.12.1.10 ZOGNetServer.BuildSubnets

Routine BuildSubnets does what its name implies: it creates a list (really a hashed table) of all the subnets in file :zognet>subnet.index using EXPORTed procedure ZOGNetServer.CrSnRecord. This table is created in the EXPORTed array, Subnets.

Basically, this procedure first initializes the array to all NILS. Then it reads one line of the file at a time, creates the entry into the Subnets array with a call to CrSnRecord, and finally adds the primary and secondary node information to the SubnetTyp data structure (q.v.)

It also reads file ZogNetFile (:zognet>zognet.setup) into the EXPORTed boolean, Only_Zognet. Finally, it reads file SecDefFile (:zognet>Sec.Default) into EXPORTed integer DefSecCnt, the default number of secondary copies of subnets to maintain, and into EXPORTed integer array, DefSecNodes[i], the Server Node number for each secondary copy of subnet to maintain.

3.1.13. NetHandl.IniNetHandl

Initialize system globals use in frame manipulation and network communication.

- Allocate memory in the default data segment for the local frame header buffer, FHBP, with "new"
- Allocate memory for the exported frame body buffer, FBP
- Create a local frame header record, FHP, with BaseLib.CrFHP
- Create a local frame record, FPX, with NetMakeDel.CrFP
- Set all error markers, exported booleans RdSig and WrSig to false
- Clear file name strings, exported strings InFileName and OutFileName
- Load the name and node number of the current (local) network node into exported variable, ThisName and local variable, ThisNode, using ZAccessProcs.GetCurNode
- Set the local agent-in-process boolean, IsAgent, to false

3.1.14. ZParse.GetRemArg

3.1.15. ZEint.EIntReady

3.1.16. ZLogFile.LogInit

Procedure LogInit creates a header for the ZOG Log file, :zog>log>zog.log. This header identifies the file as the log file, and identifies the version number of the running ZOG, notes that initialization of ZOG has been completed and identifies the current user's name, machine name, time and date.

Finally, the local boolean, ZOGInited is set to true, so that other *noteable* events in ZOG, typically entries made to the User Display, can be logged via procedures in ZLogFile.

3.1.17. ZInitExit.SetUpFrames

The final step of initialization is to determine the correct frames to display in the top and bottom windows, to read them into the window frame record structures and display them. Procedure SetUpFrames does just that, doing the top frame first, then the bottom frame. The top and bottom frame are respectively, a concatenation of the user name with Top1 and Mail1 (i.e. for user zog ZogTop1 and ZogMail1). The process used to do this follows:

- { concatenate user name with Top1 }
- ZWind.RdFWind { Read top frame into window frame record }
- ZWind.DspWind { Display the frame in the top window }
- ZWind.XChange { Make other window the current window }
- { concatenate user name with Mail1 }
- ZWind.RdFWind { Read bottom frame into window frame record }
- ZWind.DspWind { Display the frame in the bottom window }

4. Basic ZOG System flow

Once ZOG has initialized itself, the basic ZOG cycle starts up and is repeated interminably, until interrupted via the Control-C or Control-Shift-C exception handlers within ZOG. Almost all ZOG processing is invoked through this cycle and in particular, through the selection processing mechanism. (The only other processing which can occur within the ZOG system is in response to remote requests processed via the EtherNet interrupt exception handlers nested within the ZOG module.)

4.1. Selection processing mechanism

The (command intepreter) cycle consists of two sequential steps:

Get a selection character from the user (ZSel.ReturnSel)

Process the selection character (ZSel.OutS)

4.1.1. ZSel.ReturnSel

The sole purpose, from the basic ZOG point of view, of this procedure is to obtain a selection character (of an option, local pad or global pad) from the user, and return it for the appropriate processing. This character might have been selected by a *mouse event* (the clicking of one of the buttons on the bitpad puck or mouse), or by a *keyboard event* (the typing of a key on the PERQ keyboard). The implementation of this get-a-character process, however, is a bit more complicated than it would ordinarily seem, due to the inclusion within ZOG of two ancillary mechanisms:

Incremental Display mechanism (Module incDisp)

Alternate Next Frame selection mechanism (Slot Frames)

4.1.1.1 Incremental Display:

Module IncDisp is used for monitoring the state of a frame designated to be automatically updated as it is changed, presumably by some remote user(s). When a frame is re-displayed via this mechanism, the changes that have been made are highlighted in reverse video. The polling for this mechanism which determines when a frame re-display might be necessary is handled in ZSel.Return and the ZCanvas routine RdTKeyZOG.

The incremental display mechanism provides for a timed re-display of a frame. Every user-specified time interval, the frame header is examined. If it has a different version number/modification time than the frame currently being displayed on the screen, the new version is read, and displayed, with all changed being highlighted in reverse video.

This was developed in response to AirPlan needs, where the output frame is constantly being updated by the AirPlan expert system. Observers needed a mechanism to automatically be able to see these updates on the output frame.

The mechanism is invoked in procedure ZSel.ReturnSel when the mechanism is "turned on," usually via the ↑ai action. See Module IncDisp p. 46 for details of the implementation.

4.1.1.2 Alternate Next Frames (Slot Frames)

The alternate next frame mechanism was developed for use with environment frames for agents (q.v.). Each slot (i.e., an option that the user is to modify to provide an argument or parameter to an agent) on an environment frame has an associated *slot frame* linked to it. This slot frame contains information about the slot: type, default value, prompt (for SLED) and help information. It was determined that the slot frames should be "hidden" from most ZOG users. Thus, the alternative next frame mechanism was designed and implemented.

Rather than being stored in the *next frame* field of the selection record, the alternate next frame ID is stored in the selection's expansion area, preceded by the string "FrameID: ". This alternate next frame is accessed by typing an "escape" or "INS" character, followed by the selection character itself. The fact that there is a next frame is "hidden" because the parent selection still displays the "-" character after its selection character, indicating that there is no frameid in the next frame field of the selection.

4.1.1.3 Operation of ReturnSel

4.1.1.3.1 Initialize booleans EscSel to false and StRspSig to true;

4.1.1.3.2 If collecting statistics, end the system response time here;

4.1.1.3.3 Get a "Canvas event" (keyboard or mouse) loop:

Module IncDisp is used for monitoring the state of a frame designated to be automatically updated as it is changed, presumably by some remote user(s). When a frame is re-displayed via this mechanism, the changes that have been made are highlighted in reverse video. The polling for this mechanism which determines when a frame re-display might be necessary is handled in ZSel.Return and the ZCanvas routine RdTKeyZOG.

- If incremental display is on, then only poll for a canvas event until the incremental display interval has expired, (ZCanvas.RdTKeyZOG), then redisplay the frame, if necessary (IncDisp.UpdateIncDisp); otherwise, just get the canvas event without time out (ZCanvas.RdKeyZOG);

- If the canvas event is a mouse movement to other window, make the new window the current window (ZWind.XChange) and repeat loop; otherwise, check to see if it's a change to/from full window user display, and do so as required (ZUser.DspUsrDsp or ZUser.DspWind) and repeat loop; otherwise, if a legitimate character was obtained, exit the get canvas event loop; otherwise, beep and repeat the loop.

4.1.1.3.4 If collecting statistics, record statistics for the canvas event by marking the end and beginning of a user response cycle, and beginning a new system response cycle;

4.1.1.3.5 If Mouse Event, try to translate it into a selection character; otherwise, character is the one the user typed;

4.1.1.3.6 If the character is an "INS" (Escape) (presumably for alternate next frame selection), go back to the get canvas event loop to get the actual selection character;

4.1.1.3.7 Return the character to the calling routine.

4.1.2. ZSel.OutS

OutS uses the character returned to ZOG by ReturnSel to initiate some activity. In one way or another, the processing of this character will result in a ZOG action being invoked, be it going to a frame pointed to by a selection, a global pad action, a frame or selection action, or a user-input action command string.

All user-selected activity within ZOG is invoked directly or indirectly through the action command interpreting mechanism contained within Module ZAction. For a list of actions by control character see p. 37.

4.1.2.1 Operation of OutS

The function of OutS is to process the user-input character(s) or mouse button click. This may result in several actions taking place.

4.1.2.1.1 If the character is a ZOG Action Language Control character, get the rest of the action language string, and process it (ZAction.GAction). This is the mechanism by which users can directly invoke actions from the keyboard.

4.1.2.1.2 ZAction.GAction

Procedure GAction is the procedure invoked when a user directly inputs a ZOG action command via the keyboard. Simply entering a control character will cause this procedure to be invoked, via ZSel.OutS.

- Display the user-input control character
- Get the next character from the user and display it
- Use nested case statements to determine which, if any, appropriate next characters, strings or filenames to get from the user
- Get the appropriate optional strings, as needed, via specialized local routines GetChar, GetInt, GetActStr, GetActInt, or GetFilStr
- Clear the User Display Line of the text just input
- Process the action string via Z<ControlChar>Action.Cntl<ControlChar>Sel (i.e., for a control-B action, invoke ZBAction.CntlBSel), except for Control-F (invoke Control-A Action processor), Control-H and Control-U (do the appropriate functioning for backspace and OOPS), and Control-L (clear the user display)

4.1.2.1.3 Otherwise, try to obtain a pointer to the selection whose selection character matches the passed character (ZSel.GetSel).

4.1.2.1.4 ZSel.GetSel

Given a selection character, procedure GetSel returns a pointer to the current frame's selection whose selection character matches it. It examines selections in Option, Local Pad, then Global Pad order. It does so as follows:

- Invoke routine NetOption.GOptF on the current frame to try to find an option with the matching selection character. If successful, return the pointer to that option.
- Invoke routine NetOption.GPadF on the current frame to try to find a local pad with the matching selection character. If successful, return the pointer to that local pad.
- Invoke routine NetOption.GOptF on the Global Pads Frame to try to find a global pad with the matching selection character. If successful, return the pointer to that global pad.
- If none of these searches are successful, return false.

4.1.2.1.5 If the character referred to a valid selection (option, local pad or global pad), then process it by passing a pointer to the selection to ZSel.EvalSel; otherwise, beep.

4.1.2.1.6 ZSel.EvalSel

Procedure EvalSel is the main procedure by which normal selection processing takes place.

4.1.2.1.6.1 Invoke ZActUtils.GetAction to obtain a pointer to the current frame's Exit Action (if any), then again to obtain a pointer to the selected item's Selection Action (if any)

The operation of GetAction is simple: copy the action text from the selection pointer passed to it into a separate, returned, Frame String pointer.

- For every action string attached to the passed selection pointer, create a new (frame) string pointer with NetMakeDel.CrFsP
- Use NetInsert.InsEFsL to Insert this new action (Frame) string pointer at the End of the List pointed to by local pointer variable, ActP
- Return ActP

4.1.2.1.6.2 Obtain the selection's Next Frame Id, if any. This section will obtain the "hidden" next frame in the selection's expansion area if it exists; otherwise, the selection's normal next frame field is used.

4.1.2.1.6.3 If there is a next frame, first process the current frame's Exit Action via ZActUtils.DoAction. Then, save the current frame on the backup list via ZBack.SavBack. Finally, read the next frame into the current window's window structure (Does NOT display the frame at this time).

4.1.2.1.6.4 If there is a selection action, process it, via ZActUtils.DoAction.

DoAction merely records statistics, depending on the type (Frame action, Selection action or Exit action) of action it is to execute. Then, it calls ZAction.ProAction with the action string pointer that was passed to it.

4.1.2.1.6.4.1 If Statistics gathering is turned on, increment the appropriate action string execution counter

4.1.2.1.6.4.2 If it's an Exit action, null out the Frame-Displayed item in the current window frame record (WPt.DisFrameId), so that an infinite loop will not occur (which could happen on the return back to ZSel.DspFrame in ZSel.OutS.)

4.1.2.1.6.4.3 Invoke ZAction.ProAction to process the action passed via an action string pointer

Procedure ProAction processes the passed action string pointer, ActP, going through and extracting, then processing each action string in the string list one at a time.

As long as the action string pointer, ActP is not NIL do:

- Invoke ZAction.ProActStr with the text string pointed to by ActP
- Save ActP, then let ActP point to the next string in the list
- Make ActP pointer to the previous string = NIL, to eliminate the already processed dynamic string variables
- Invoke NetMakeDel.SavFsP to save the string pointer for later use within ZOG

4.1.2.1.6.4.3.1 Invoke ZAction.ProActStr with the text string pointed to by ActP

Procedure ProActStr receives as its argument a string, presumably containing ZOG action commands. It parses this string for individual action commands, then dispatches them to be interpreted, as follows.

For as long as there is text in the argument string do:

4.1.2.1.6.4.3.1.1 Load a character at a time into the local action string variable, ActTxt, until the next character is a control character (one of Control-A, Control-B, Control-D or Control-E);

4.1.2.1.6.4.3.1.2 Invoke ZAction.XAction to execute this individual action.

Procedure XAction parses the string passed to it into a control character, a command character, and possibly an optional subsequent string. It then passes control to the appropriate dispatching procedure in the correct Module.

4.1.2.1.6.4.3.1.2.1 Based on the length of the input string, get the correct action optional string argument

4.1.2.1.6.4.3.1.2.2 Via a case statement, based on the first character of the string (the control character), dispatch to the appropriate Cnt<ControlChar>Sel procedure to invoke the correct action.

NOTE: if the first character is NOT a control character, the default action mechanism is to just print the action string in the user display.

4.1.2.1.6.4.3.1.2.3 Action procedure invoking

After the action string is passed to routine ZAction.XAction, it is interpreted according to the first (control) character of the string and then dispatched for execution to one of the following modules. NOTE: Global Pads actions are indicated with the global pads name in quotes.

- ZAAction.CntlASel - Control-A Actions
- ZBAction.CntlBSel - Control-B Actions
- ZDAction.CntlDSel - Control-D Actions
- ZEAction.CntlESel - Control-E Actions
- ZUser.ClrUsrDsp - Control-L Action
- ZVideo.CntlVSel - Control-V Actions
- ZIO.WrStrLn - Default; just print action string

4.1.2.1.6.4.3.1.2.3.1 Control-A Actions : Module ZAAction

Control-A actions allow users to move around within and between ZOG subnets. Many of these actions are "hidden" as action strings attached to the options of the global pads frames (thus becoming Global Pads for all other ZOG Frames).

4.1.2.1.6.4.3.1.2.3.1.1 Frame/Subnet Traversal Actions

↑A0	ZAAction.TopOfSubnet - Go to frame 1 of current subnet. Usually the action of the *. top Local Pad.
↑Ab	ZBack.XGoBack - "Back" - Pop a frame off the backup list and display it in the current window.
↑Af	ZAAction.GoToFrame - "Find" - Go to the Find Agent's environment frame.
↑Ag	ZAAction.GoToFrame - "GoTo" - Read and display the user- supplied frame in the current window.
↑An	ZBack.XNext - "Next" - Go to next frame on parent frame's option list.
↑Av	ZBack.XPrev - "Prev" - Go to previous frame on parent frame's option list.

4.1.2.1.6.4.3.1.2.3.1.2 Display Controlling Actions

↑A #	ZAAction.SetUserLine - Set the user display line to be the user- specified line (default = 23).
↑A +	ZAAction.ExpandBig - Expand the current frame display window to BigFrame (80 x 49) size. (Also, ↑F +)
↑A-	ZAAction.ShrinkBig - Contract the current frame display window to normal (80 x 24) size. (Also, ↑F-)

↑Ad	ZAAction.DispFr - "Disp" - Read and display the current window's frame.
↑Ai	ZAAction.IncDisp - Set Incremental Display interval time via IncDisp.SetIncDisp. This procedure sets EXPORTed integer IncDispTime and EXPORTed Boolean, IncDispSig.
↑Aj	ZWind.XChange; - "Jump" - Display current frame in other ZAAction.GoToFrame window, if there is an other window.
↑Au	ZUser.DspUsrDsp - Change to full window, and display all user messages therein.
↑Aw	ZAAction.ChgWind - "Win" - Change to other window if possible.
↑A<	ZDisplay.ClrLine - Clear the current line in the window.

4.1.2.1.6.4.3.1.2.3.1.3 Backup List Handling Actions

These procedures maintain the backup frames list. The actions to go back, go to next and go to previous frames are listed in the Net Traversal frame, above.

↑A$	ZBack.XRetBack - "Ret" - Display the Backup List pseudo-frame so users can directly access backed up frames from this pseudo-frame.
↑Ac	ZBack.XClrBack - Clear the list of Backed up frames. (Also clears Marked frame list.)
↑Ap	ZAAction.Pop - Pops the backup list back up to the specified frame via ZBack.XPopBack (and ZBack.DelBack and ZBack.ErBack).

4.1.2.1.6.4.3.1.2.3.1.4 Net Traversal

4.1.2.1.6.4.3.1.2.3.1.5 Mark List Handling Actions

The Frame "Marking" routines were just recently added to PERQ ZOG, with the inclusion of Module ZMark. ZMark maintains a list exactly like the frame backup list which Module ZBack maintains. When users "mark" frames, they are merely added to the Mark list. These frames can then be revisited.

↑A%	ZMark.XRetMark - Display a Mark list pseudo-frame, so user can access list of Marked frames.
↑A	ZAAction.MarkFr - Add the specified frame to the Mark list via ZMark.SavMark and ZMark.InsMark.
↑Ac	ZMark.XClrMark - Clear the entire list of Marked frames.
↑Am	ZAAction.MarkFr - Add the frame displayed in the current window to the Mark list.
↑Aq	ZAAction.RemMark - Remove the specified frame from the Marked frame list via ZMark.DelMark.
↑Ar	ZMark.XLastMark - Go to the last (i.e., most recent frame on the Mark list) in the current window.

4.1.2.1.6.4.3.1.2.3.1.6 Top, Bottom and Help Frame Accessing

↑A1	ZAAction.SetInitFr - Resets ZOG Top Frame to user-specified frameid
↑A2	ZAAction.SetFrFile - Resets ZOG Bottom Frame to user-specified frameid
↑A3	ZAAction.SetFrFile - Resets ZOG default Help Frame to user-specified frameid
↑Ae	ZAAction.GoToFileFrame - "Help" - Displays standard help frame, (:zognet>help.frame)
↑Ak	ZAAction.GoToFileFrame - ("Bottom") - Go to the initial frame displayed in the bottom window (:zognet>bottom.frame)
↑Ao	ZAAction.TopFrOfNet - "Top" - Go to the initial frame displayed in the top window (:zognet>top.frame)

4.1.2.1.6.4.3.1.2.3.1.7 Actions Affecting Global Pads Frames

↑A&	ZAAction.SetAltGPads - Set the Alternate Global Pads frame associated with the frame displayed in the current window to be the specified frame.
↑A*	ZAAction.SetDefGPads - Set the ZOG-wide Default Global Pad frame, contained in global string, DefGPad, EXPORTed from Module NetMakeDel, to be the specified frame.
↑A/	NetHandl.RdF - Read in and display the alternate global pads set of the frame displayed in the current window, if it exists.
↑A>	ZDisplay.DGPads - Display the current frame's Global Pads set.
↑A\	NetHandl.RdF - Read in and display the standard global pads set of the frame displayed in the current window.

4.1.2.1.6.4.3.1.2.3.1.8 Miscellaneous Actions

↑Aa	(Not Implemented - Used to be the accessors handling action)
↑At	ZParse.ProSwitches - Toggle specified global tracing booleans (declared in Module ZTrace) to enable/ disable certain debug tracing.
↑Az	ZAAction.ZOGSleep - Put ZOG to sleep for specified number of seconds; i.e., sit in a tight loop until the specified number of seconds have elapsed.

4.1.2.1.6.4.3.1.2.3.2 Control-B Actions : Module ZBAction

The Control-B actions typically deal with input to and output from ZOG. These actions fall under the following categories:

4.1.2.1.6.4.3.1.2.3.2.1 Canvas Display Controlling actions

↑B+	ZCanvas.SetCanvFunc - Select black or white background for PERQ screen, along with mouse pointer display mode, via POS-supplied procedure, IO – Others.IOSetFunction. See PERQ manual for codes.
↑B↑	ZCanvas.SetCanvPtr - Select the image for the mouse pointer, as supplied in file :zognet>ZOG.animate. At the present time, only two images are available. See IO – Others.IOLoadCursor.

↑Bd	ZBAction.PosCursor - Position the cursor to the column and row specified in the command string.
↑Bj	ZCanvas.SetAltCursor - Sets the alternate cursor (for highlighting selections or text) to the specified type.

4.1.2.1.6.4.3.1.2.3.2.2 Opening and Closing of special ZOG input and output files

These actions make use of global text variables, NetInFile and NetOutFile, and booleans, RdSig and WrSig, EXPORTed from NetHandl, to open and close files for reading or writing from within ZOG. In particular, the open file routines must be done prior to any of the Read/Write ZBH file actions. Typically, these are stacked in the action string for the specific options so that they are executed before the file I/O action is actually invoked.

↑Bc	ZBAction.ClsFile - Close the system file(s) which is open: if RdSig is true, then close NetInFile; if WrSig is true, then close NetOutFile. Set both booleans to false.
↑Bi	ZBAction.OpnInFile - Open user-specified input filename as text file NetInFile; set Boolean RdSig to true.
↑Bo	ZBAction.OpnOutFile - Open user-specified output filename as text file NetOutFile; set Boolean WrSig to true.

4.1.2.1.6.4.3.1.2.3.2.3 Agent and Shell Utility Invocation

These two actions provide the entry points for users to execute agents and to perform shell utility functions (such as typing a file, obtaining a directory listing, changing the current path, etc.) The routines called simply parse the action string for the name of the agent/shell utility to be executed, then dispatches to the appropriate <AgentorShellModule>.<AgentorShellProcedure>.

↑Ba	ZAgent.CallAgent - Parse argument string into top agent frame id, Agent's environement frame and agent name. Based on agent's name, dispatch to appropriate procedure <AgentName>.Do<AgentName>.
↑Bq	ZShell.CallShell - Do preliminary parsing of input string, setup, and ZCanvas.ClearCanvas to clear current window. Call ZXShell.ZShell to parse shell command name and invoke the appropriate procedure, Z<ShellName>.Do<ShellName>.

4.1.2.1.6.4.3.1.2.3.2.4 Miscellaneous actions

↑B.	ZDisplay.Clear - Clear the current window
↑B?	ZBAction.Comment - Display Frame comment in user display line
↑B;	- Set the EXPORTed Boolean, CommentSig, to true (This boolean is no longer used within ZOG)
↑Bn	- Set the IMPORTed Boolean, SigNoClear, to true (This boolean is no longer used within ZOG)
↑Bs	ZBAction.PrintChar - Print the specified character in front of the specified option
↑Bt	ZBAction.ChangeTerm - Change external display terminal characteristics

↑Be	Except.Raise - Performs an automatic exiting of ZOG by setting Boolean, ForcedLogOut, to true, then Raising the Control-Shift-C exception, CtlShftC.

4.1.2.1.6.4.3.1.2.3.3 Control-D Actions : Module ZDAction

With the exception of the the 4 videodisc controlling actions, most of the Control-D actions involve frame modification, including the invocation of the ZOG editors. Since there are 30 Control-D actions, they are listed below by functional category.

1. Videodisc Controlling Actions
2. Frame and Subnet Protection Field Modifying Actions
3. Frame Owner String Modifying Actions
4. Subnet Creation and Deletion (Clearing) Actions
5. Frame Creation and Erasing Actions
6. Editor Invoking Actions
7. ZBH Subnet File I/O Actions
8. Miscellaneous Actions

4.1.2.1.6.4.3.1.2.3.3.1 Videodisc Controlling Actions

These actions, usually pre-installed as option or pad action strings, allow users to use the videodisc player as a means of displaying additional, perhaps graphical or pictorial, supplementary information. This feature has been extensively incorporated into the ship's weapons and aircraft elevator manuals for illustrative and educational purposes. These ↑D videodisc actions are an upper-level subset of the videodisc commands. See Module ZVideo.

↑D%	ZVideo.VideoS - Display the specified videodisc frame on the videodisc player's monitor, via UEI.VideoFrame.
↑D%	ZVideo.VideoR - Display the specified sequence of frames (clip) on the videodisc player's monitor, via UEI.VideoRange.
↑D	ZVideo.InitV - Initialize the RS-232 port for PERQ control over the videodisc player, via UEI.VideoInit.
↑Dq	ZVideo.QuitV - Secure the videodisc and discontinue PERQ/ZOG control over the player over the RS-232 port, via UEI.VideoQuit.

4.1.2.1.6.4.3.1.2.3.3.2 Frame and Subnet Protection Modifying Actions

These Actions modify the protection field in individual frames, or groups of frames. It is unclear why the numbers for the ↑D1 through ↑D6 actions bear no direct correspondence to the protection levels as defined in NetDefs.

↑D<1..6>	ZDAction.FProtect - Set Protection field of specified frame to be the following 1 = Read Protection; 2 = Write Protection; 3,4,6 = No Protection; 5 = Modification Prote[illegible]on
↑Da	ZDAction.SetDefProt - Set the default protection of all new frames to be created by current user, DefProt, a ProtTyp variable, EXPORTed from BaseLib.
↑Dp	ZDAction.SnProtect - Set the protection field of each frame in the specified subnet to the specified level of protection.

4.1.2.1.6.4.3.1.2.3.3.3 Frame Owner String Modifying Actions

These actions allow users to modify (add or delete) the list of owners associated with a given frame.

↑D7	ZDAction.AddOwner - Adds the specified owner to the given frame's owner list.
↑D8	ZDAction.RemOwner - Removes the specified owner from the given frame's owner list.
↑D =	ZDAction.AddOwner, ZDAction.RemOwner - Changes owner of frame displayed in current window, first adding, then deleting an owner, as per user input in the user display line.

4.1.2.1.6.4.3.1.2.3.3.4 Subnet Creation and Deletion (Clearing) Actions

These two actions allow users to create and delete (really, clear all the frames from) the specified subnet.

↑Dc	ZDAction.CrSubnet - Creates a subnet with NetHandl.CrSnSec call, then creates frame <SubnetName>0 in that new subnet with NetHandl.CrFr.
↑Dx	ZDAction.ClearSn - Clears (removes all frames from) the specified subnet via NetHandl.ClrSn. After this is accomplished, the subnet still exists (in subnet.index), but has no frames.

4.1.2.1.6.4.3.1.2.3.3.5 Frame Creation and Erasing Actions

These actions allow users to create and erase frames.

↑Dd	ZDAction.EraseFrame - Erase (Delete) specified frame from subnet, via NetHandl.ErF. If this is successful, the links back from the deleted frame's parent's options and local pads are removed.
↑Di	ZDAction.CrFrame - Create the specified frame, if it does not already exist. If the subnet for the frame does not exist, create the subnet as well, via NetHandl.CrSnSec. In any case, control is passed to ZCrFrame. CrF – CtlDi for the actual creation of the frame.

4.1.2.1.6.4.3.1.2.3.3.6 Editor Invoking Actions

These actions allow users to invoke either of the two ZOG Editors.

↑De	ZDAction.EditFrame - "Edit" - Invoke the ZOG Frame Editor, ZED, on the frame specified by the user. This calls procedure ZEdit.EdFr to actually perform the editing.
↑Dl	ZDAction.SlotEditFrame - "Sled" - Invoke the ZOG Slot Editor, SLED, on the frame specified by the user. This calls procedure ZSled.SlEdFr to perform the actual slot editing.

4.1.2.1.6.4.3.1.2.3.3.7 ZBH Subnet File I/O Actions

These actions allow users to transfer subnets from ZOG accessable (primary) form (filename <SubnetName>.pri) to the ZBH (transportable) form (filename <SubnetName>.zbh) and vice versa.

↑D<	ZBHIO.RdBH - Reads in a ZBH file and creates the corresponding subnet (.pri) file, and updates subnet.index if necessary. Note - Presumes that the ↑Bi action (ZBAction.OpnInFile) has already been done to specify the ZBH file to be read.
↑Df	ZDAction.WrFrBH - Writes out the specified frame into the ZBH file format (no holes) via ZBHIO.WrFBH. Note - Assumes that the ↑Bo action (ZBAction. OpnOutFile) has already been done to open the output file for writing.
↑Ds	ZBHIO.WrSNetBH - Writes out the specified subnet into a ZBH format file. Note - Assumes that the ↑Bo action (ZBAction.OpnOutFile) has already been done.

4.1.2.1.6.4.3.1.2.3.3.8 Miscellaneous Actions

Miscellaneous Informational Actions. These actions merely print out a string on the user display line giving the user some requested information.

↑D?	ZDAction.Info - "Info" - Print a string giving the full frame header information on the specified frame and highlight any differences between the current frame and the old frame if there is one.
↑Do	ZDAction.DFrId - Overlay the specified frame in the current window
↑Dt	ZIO.WrIntLn, - Print the system time in seconds since midnight. BaseLib.GTimInt
↑Du	NetHandl.GldUsr - Print the name of the currently logged-in user
↑Dv	ZIO.WrStrLn - Print the current ZOG Version number, ZOGVersNumber, IMPORTed from ZOGVersion.
↑Dy	ZDAction.DFrId - Display the specified frame in the current window. Note that the frame's Global Pads are not displayed.

Special Features.

↑dm and ↑dn	The Playback feature
↑dz	Show stats on all Perqs
↑d$	Show stats on a single Perq

4.1.2.1.6.4.3.1.2.3.4 Control-E Actions : Module ZEAction

This module provides procedures for generating output to the Canon Laser Printer from within ZOG. The two currently implemented procedures and their command characters are:

↑Ef	ZEAction.PrintFile - Print the specified file remotely on the Canon Laser Printer.
↑Es	ZEAction.ScreenDump - "Can" - Dump the image of the current screen to the Canon Laser Printer.

4.1.2.1.6.4.3.1.2.3.5 Control-L Action

There is only one action which is invoked by Control-L. This action deletes the entire contents of the user display buffer and clears the user display window, via ZUser.ClrUsrDsp (ZUser.InitUsrDsp and ZCanvas.ClearCanvas).

4.1.2.1.6.4.3.1.2.3.6 Control-V Actions : Module ZVideo

Module ZVideo contains all the Videodisc control actions available to ZOG. It contains commands to initialize the video disk player, to show a specific frame or sequence of frames and to search for a selected frame on the video disk.

The video disk player used by the USS CARL VINSON can be controlled by a wired remote control pad or through an RS-232 port. Module UEI contains and exports the procedures to send the appropriate sequences of command characters to the videodisc player to initiate various operations. UEI (Universal External Inerface) is imported by Module ZVideo.

4.1.2.1.6.4.3.1.2.3.6.1 Frame and Sequence Display Actions

↑V%	ZVideo.VideoS - Display the specified video disk frame on the video disk player's monitor, via UEI.VideoFrame
↑V%	ZVideo.VideoR - Display the specified sequence of frames (clip) on the video disk player's monitor, via UEI.VideoRange
↑Vf	UEI.VDFrameDisplay - Toggle the display of the video frame number on the monitor screen
↑Vh	ZVideo.Search - Search for and display the specified frame. Calls UEI.VDSearch.
↑Vl	UEI.VDSlowForward - Slow forward through frames
↑Vm	UEI.VDSlowReverse - Slow reverse through frames
↑Vn	UEI.VDScanForward - Fast forward to scan video frames
↑Vo	UEI.VDScanReverse - Fast reverse to scan video frames
↑Vs	UEI.VDStop - Stops the play of the video disk
↑Vy	UEI.VDPlay - Plays video disk in normal mode from current video frame on until stopped

4.1.2.1.6.4.3.1.2.3.6.2 Video Player Programming Actions

↑Vc	UEI.VDClr - Clears the storage registers of the videodisc player
↑Ve	UEI.VDEndProgram - Cues the videodisc to mark the end of a user-written program.
↑Vr	ZVideo.Recall - Recalls a frame number from the user- specified videodisc player register.
↑Vt	UEI.VDStore - Stores a frame number into the specified videodisc player register.
↑Vw	ZVideo.WriteProgram - Allows the user to create a program of frames to be displayed on the videodisc player via UEI.VDWriteProgram.
↑Vx	ZVideo.RunProgram - Runs the user-created program on the videodisc player via UEI.VDRunProgram.

4.1.2.1.6.4.3.1.2.3.6.3 Video Disk Initialization and Shutdown Actions

↑Vi	ZVideo.InitV - Initialize the RS-232 port for PERQ control over the videodisc player, via UEI.VideoInit
↑Vj	UEI.VDReject - Causes the videodisc heads to be retracted and the videodisc itself to be ejected from the player
↑Vz	ZVideo.QuitV - Secure the videodisc and discontinue PERQ/ZOG control over the player over the RS-232 port, via UEI.VideoQuit

4.1.2.1.6.4.3.1.2.3.6.4 Miscellaneous Video Disk Actions

↑V1	UEI.VDAudio1 - Toggles the first audio channel on the videodisc player on and off
↑V2	UEI.VDAudio2 - Toggles the second audio channel on the videodisc player on and off
↑Va	ZVideo.AutoStop - Cause the videodisc player to play forward until the specified video frame is reached. Calls UEI.VDAutoStop.
↑Vp	UEI.VDStepForward - Step the videodisc player forward one frame at a time
↑Vq	UEI.VDStepReverse - Step the videodisc player reverse one frame at a time

4.1.2.1.6.4.3.1.2.3.7 ZIO.WrStrLn - Default; just print action string

4.1.2.1.6.4.3.2 Save ActP, then let ActP point to the next string in the list

4.1.2.1.6.4.3.3 Make ActP pointer to the previous string = NIL, to eliminate the already processed dynamic string variables

4.1.2.1.6.4.3.4 Invoke NetMakeDel.SavFsP to save the string pointer for later use within ZOG

4.1.2.1.6.5 If there was no next frame AND no selection action, do top-down frame creation.

Top-Down Frame Creation is the normal route by which new frames are created with links to a parent frame's selections. Whenever a selection is made by the user which has no next frame and no selection action, the user is asked if he/she wishes to create a new frame. If the answer is yes, the new frame is created and displayed in the current window, and the user is placed in the ZOG Editor, ZED.

- Invoke ZCrFrame.CreateF to query the user, and if appropriate, create the new frame, returning the new frame's frame id
- If a new frame was, in fact, created, save the current frame on the Backup list via ZBack.SavBack
- Read in the newly created frame into the window structure with ZWind. RdFWind; then use ZWind.DspWind to display this frame (now stored in the current window's frame structure) in the window

4.1.2.1.7 If a new frame has been loaded into the window record as a result of the above, then display it in the current window and execute any frame action associated with the new frame, if applicable (ZSel.DspFrame).

4.2. I/O to the Screen display

The display modules control ZOG's interaction with the PERQ screen, keyboard and bit pad puck (and also the auxilliary CRT Display). They cause text and ZOG's window graphics to be displayed on the PERQ display unit and send text out to the auxilliary display. They also obtain input from the keyboard and the bit pad puck. The display I/O procedures are called from throughout ZOG.

The 7 functionally distinct display modules are:

4.2.1. Module IncDisp -automatically updating displayof a frame periodically

Module IncDisp is used for monitoring the state of a frame designated to be automatically updated as it is changed, presumably by some remote user(s). When a frame is re-displayed via this mechanism, the changes that have been made are highlighted in reverse video. The polling for this mechanism which determines when a frame re-display might be necessary is handled in ZSel.Return and the ZCanvas routine RdTKeyZOG.

InitIncDisp	Create scratch record and initialize incremental display (local boolean, IncDispSig) to off.

SetIncDisp	Given an update timing interval, set local variable IncDispTime and IncDispSig accordingly, to turn incremental display on or off, as requested.
UpdateIncDisp ·	Redisplay frame with highlighted changes if frame has changed and if display timing interval has expired.
SavIncDisp	Save the frame ID of the frame displayed in the current window and mark it as not having changed.

4.2.2. Module ZWind

The ZWind Module performs the task of managing the ZOG frames in their separate windows.

XChange	Change current window (to the other window) (raw action).
SetWind	Set the name of the frame, global pads frame and the selection character used to get there into current window record.
DspWind	(Re)Display the frame in the current window. This is also used to leave the full-screen user display.
RdFWind	Read a frame into the current window frame record. In general, a backup stack entry should be pushed before this is called.
OpnWind	Open the frame in the current window (lock the record and read it in) (presumably for subsequent modification).
ReInitWind	Reinitialize window records for newly logged-in user.

4.2.3. Module ZIO

The ZIO module contains the routines for writing message to the user. These messages appear on the User Display Line. Whenever a new message is written to the User Display Line, previous lines are automatically scrolled into the User Display Window. Programmers should always use these routines, not PASCAL write(ln)s whenever possible in order to scroll the messages.

WrStr<ln>	Write a string (and cr/lf)
WrInt<ln>	Write an integer (and cr/lf)
WrChr<ln>	Write a character (and (cr/lf)
WrReal<ln>	Write a real (and cr/lf)
WrBool<ln>	Write a boolean (and cr/lf)
WrAir<ln>	Write an airplan message (and cr/lf)
SendChar	Write a character to the RS232 port (i.e., the auxilliary terminal or video disk)
SendString	Write a string to the RS232 port

4.2.4. Module ZUser

This module provides byte (character) stream output to both these windows (the *small* User Display Window beneath the two frame windows, and the Full Screen User Display Window, which can be superimposed over the entire PERQ display screen), which is done through procedure WrUsrDsp. The rest of the routines manipulate the two User Display Windows.

WrUsrDsp	Write a character to the User Display.
ClrUsrDsp	Clear the User Display Line and Window. (↑L action) (This clears all the user display data structures)
DspUsrDsp	Change windows to the Full Screen User Display window, and display the last screen-full of error messages. (↑Au)
DspFUsrDsp	Display the last User Display Window-full of error messages in the (small) User Display Window.

4.2.5. Module ZDisplay

The ZDisplay module processes ZOG frames and breaks them into pieces which can be passed to the lower level utilities in ZCanvas. It also provides more complex display utilities.

Clear	Clear the current window
DspPos	Position the cursor to specified location in the frame display
DspEnd	Position the cursor to (UsrDispLine,1)
ClrEOLn	Clear from current position to end-of-line
ClrLine	Clear the entire current line
DspF	Clear the current window and display specified frame
DspF1	Display specified frame without clearing
DGPads	Display Global Pads from the specified GPads frame
DspSelF	Display Selection records
DspLPads	Display Local Pads records
DspFsP	Display a Frame String record (i.e., item text)
DspStar	Mark the selection with an asterisk
DspCxt	Display a context string ('edit','second',etc.) to the left of the frame id (upper RH corner)

4.2.6. Module ZCanvas

Module ZCanvas, along with ZCanvUtils, provide the lowest level of routines to interface direclty between ZOG and the POS screen utilities. NOTE: ZCanvas also manages I/O to the the auxilliary terminal if necessary.

The routines ZCanvas provides are:

4.2.6.1 Canvas (i.e. window) manipulation routines

ChangeCanvas	Go from the current window to another window. If both windows contain ZOG frames, the new window is highlighted (i.e., outlined with a reverse video frame containing the ZOG title).
TitleCanvas	Change the Title in the current window to the one supplied
ClearCanvas	Clear the current window
SetCanvas	Set the internal variables for the current window
SetCanvPtr	Select an image for the mouse pointer from among the ones available (currently, a solid and a hollow arrow)
SetCanvFunc	Select black or white background, and the mouse pointer display mode (XOR or Replace)

4.2.6.2 Display adjusting routines

SetPtrCh	Set the position of the mouse pointer. Also moves the alternate cursor correspondingly.
SetPosCh	Set the position of the cursor
GetPosCh	Get the position of the Mouse
CursorOn	Turn on the cursor
CursorOff	Turn off the cursor
IsCursorOn	Is the cursor on ?

4.2.6.3 Alternate cursor routines

The *alternate cursor* in ZOG refers to one of the available displays that can be attached to the mouse pointer, and moved with it. These include horizontal or vertical lines or reverse video boxes (the full width/height of the window) and box/reverse video highlighting of a frame's selections. (See frame ZOG8 for a full list). The change from one alternate cursor to another can be invoked via the ↑bj action, with a numeric argument.

SetAltCursor	Select which alternate cursor is to be used. A 0 argument will turn off the alternate cursor altogether.
DispAltCursor	Erase the current alternate cursor, and display a new one at the specified coordinates.
IsAltCursor	Returns the current state of the alternate cursor.
LineCanvas	Draws a line between specified points, for alt. cursor.
BoxCanvas	Draws a box in the window with the specified corners, for surrounding options alt. cursor.

4.2.6.4 Output routines

These procedures will "print" characters on the PERQ screen, and also on the auxilliary display unit (CRT), if being used.

SetChFunc	Select Replace, OR, XOR, etc. character display function
GetChFunc	Return the current character display function
PutCh	Output a character
PutStr	Output a string
PutStrLn	Output a string followed by a cr/lf
PutSubStr	Output a substring
BEEPCanvas	Cause the terminal speaker to emit a short "beep"

4.2.6.5 Input routines

These routines poll for an input canvas event. The alternative form of routines with the letter "T" include a timeout feature, that is, the routine will return a value of false if no input was received within the specified time interval. This feature is used to implement the automatic incremental display mechanism.

Rd[T]Canvas	Get next input from the user, including movement of mouse
Rd[T]KeyEv	Get next input, ignoring mouse movement; user is NOT allowed to leave current window (used in editors)
Rd[T]KeyZOG	Same, but user is allowed to leave the current window
Rd[T]Kbd	Get a character from the keyboard, ignoring mouse
RdKbdCond	Return character, or null if nothing typed
RdCanv	Detects an input event (from keyboard or mouse) and returns it in the CanvEv record.
IsMouseEv	Return true if canvas event was mouse button click

4.2.7. ZCanvUtils - Lowest level PERQ Screen interface routines

SetZOGCanvas	Initialize 5 ZOG Frame Canvases and record structures
IniCanvPtr	Read in Mouse pointer images from :zognet>zog.animate
CharToAbs	Convert normal x-y character coordinates to screen pixel (absolute) coordinates
AbsToChar	Convert pixel (absolute) coordinate points to character x-y cordinates, in the proper window
ScreenLine	Draw a line on screen connecting two absolute coordinates
DrawBox	Draw a box on screen with opposite corners given in absolute (pixel) coordinates

4.3. ZOG's Exception Handling:

Within the PERQ Operating System, POS, many events can generate an exception, that is, a system-level interrupt or trap. Simple examples of these are division by zero, disk I/O failures, bad pointers in memory, and the like. PERQ PASCAL provides a mechanism by which these exceptions can be *fielded*. Without these exception handlers, as they are called, any exception will cause the applications program to abort. There are a number of exception handlers built into the top level of ZOG to handle these system-generated exceptions. These are:

- Special Character generated exceptions
- Specific POS-generated exceptions handled by ZOG
- EtherNet Receive completed Exceptions
- All Exception
- ZOG-Defined Exceptions (LogOutZOG)

4.3.1. Special Character Exception Handling:

4.3.1.1 Control-C Exception Handler CtlC

This handler traps the user's Control-C input (which causes the Control-C exception, declared in POS file System, to be raised), and invokes local function ReLogZOG. This function "suspends" ZOG execution via ZOG.SuspZOG, which saves the current state of ZOG, then it prompts for input as to whether or not the user wishes to log out of ZOG. If the user, in fact, wishes to log off, then procedure ZInitExit.LogOffZOG is called to perform the logoff, dump some statistics, and invoke ZLogin.DoZOGLogIn to display and request a new user name (and password.)

4.3.1.2 Control-Shift-C Exception Handler CtlShftC

This handler will trap the Control-Shift-C exception (declared in System.pas) raised by POS when a Control-Shift-C is received from the keyboard. Normally, this exception results in an abort of the current process. Within ZOG, however, this is the mechanism by which a user terminates execution of ZOG and returns to the POS shell.

CtlShftC after clearing the exception, invokes local function ReInitZOG, which, in turn, suspends ZOG exection via ZOG.SuspZOG. It prompts the user for input as to whether or not an exit from ZOG is, in fact, desired. Upon receipt of a positive response, or if there was a "forced" log out via another exception, ZInitExit.ExitZOG is called to "shut down" ZOG in a controlled fashion. The canvas is cleared, ZOG is exited and control passes to the POS Shell.

An unusual feature of the Control-Shift-C exception on the PERQ is that if the exception is handled, and the handler is exited, the Control-C exception is raised! Thus, if the user responds negatively (default) to the prompt, above, the CtlShftC handler is exited, and the Control-C exception raised. As a result, the Contol-C handler (q.v.) is invoked.

4.3.1.3 Control-Shift-D Exception Handler Dump

The Dump exception (declared in POS module Except) is raised whenever the user types a control-shift-D. This exception is trapped by handler Dump in ZOG, wherein the user is asked if a procedure stack dump is desired. If so, the stack dump is printed on the screen, and control returns to ZOG.

These stack dumps are occasionally useful for debugging purposes within ZOG, since they list all the procedures currently invoked. This handler was necessary, since without it, the stack dump would have automatically been invoked, AND ZOG would have been aborted, via system module Scrounge. This did, in fact occur in earlier days, since some users interpreted the documentation for the Control-D actions as requiring a Control-SHIFT-D to invoke, with "disastrous" results.

4.3.1.4 Help Key Exception Handler HelpKey

This handler traps the HelpKey exception (declared in POS Module System), and returns the character "h", corresponding to the "help" global pad. Thus, the help key, in effect, by being trapped in this way, simply acts like any other key on the keyboard as far as ZOG is concerned.

4.3.2. Specific POS-generated exceptions handled by ZOG

- FileSystem File Not Found (FileSystem.FSNotFnd) Exception Handler, to simply handle the exception and return control to the point at which the exception was raised, to allow local processing of this situation. In actual fact, most modules in which there is a possibility of this exception being raised, declare their own specific handler for this, which takes precedence over the handler defined at the top ZOG level.

4.3.3. EtherNet Receive completed Exception Handler E10ReceiveDone

The EtherNet hardware on the PERQs will generate a hardware interrupt when a message has been received. Procedure EtherInterrupt.E10Srv handles this interrupt, and then, after some processing, loading the message into application accessible buffers, raises the exception E10ReceiveDone. This is the way that ZOG is signaled that an Ethernet message receipt has been completed.

Within the main ZOG Module, nested exception handlers (currently four deep) for this E10ReceiveDone exception have been written. This handler dispatches out the correct routine to handle the message. Typical messages are AirPlan messages to be displayed, frame read/write requests from remote PERQs, acknowledgement messages, frame headers/bodies being returned from remote PERQs in response to the local PERQ's request and occasional EtherNet probes. The AirPlan messages are handled by QueueAirOutput; all the rest are dispatched to ZNetServer.ZNetServer (q.v.) to further parse and respond to as necessary. As a signal to the user that an EtherNet interrupt has occurred, the mouse pointer is replaced by the hollow arrow (ZCanvas.SetCanvPtr).

4.3.4. All Exception

In order to prevent ZOG from aborting suddenly, leaving the disk cluttered with temporary segments, and to allow for statistics and exception recording when an abort-producing exception is raised, the All Exception handler was written and incorporated into ZOG.

This Handler "catches" any PERQ POS exceptions not specifically handled by other handlers. It does the following:

- Invoke procedure ZInitExit.ExitZOG to perform the controlled "shutdown" of ZOG, including statistics recording if they're turned on;
- Clear the Canvas, and give the user an error message;
- Invoke procedure ZDump.ZOGDump to record the stack dump (normally output to the screen with POS Module, Scrounge) in files :zog>log>exception.log and at the end of the current logging file, :zog>log>zog.log;
- Raise the ExitProgram exception, which causes the execution of ZOG to terminate.

4.3.5. ZOG-Defined Exceptions

PERQ PASCAL also provides a mechanism, via POS Module Except, for user programs to declare, and raise exceptions. Several of these are defined and handled within ZOG itself. These are:

4.3.5.1 LogOutZOG Exception Handler LogOutZOG

As of ZOG Version 28.0 this is no longer used The LogOutZOG E[illegible]n (declared in Module NetHandl) Handler provides and automatic logout machanism. It was incorporated into ZOG to accomodate the ship's request to have ZOG log itself out automatically to avoid unauthorized access to ZOG. This situation could arise if a person left his PERQ running ZOG for an extended period of time without first logging out. With that PERQ unattended, anyone could sit down and be able to access everything that the original person was privileged to.

To avoid this, procedure ZPollSnap.PollProc (which is called every RegularInterval (72000) jiffies (corresponding to a 20 minute time interval) from ZCanvas.PollCanvas) checks to see if there was any activity at all during the last SnapShot period. If not, no snapshot is t[illegible]ken, AND Exception

LogOutZOG is raised in this procedure. The exception is then handled by the LogOutZOG Exception Handler declared in ZOG itself.

4.3.5.2 AbortAgent Exception Handler ZX<A or B>Agent.AbortAgent

One feature which earlier versions of ZOG lacked was the ability to (semi-) gracefully stop the execution of an agent once it was invoked when some frame access failed. In order to install this, Exception AbortAgent was declared in Module NetHandl, which is the upper level frame access module.

This exception is raised in the NetHandl procedures whenever a frame access fails AND an agent is running (private boolean, IsAgent, in NetHandl). The exception is handled at the ZXAAgent.DoAAgent or ZXBAgent.DoBAgent procedure level, which simply exits that procedure and returns up the PASCAL stack through the action processor back to ZOG. The messages describing the problem in frame accessing which caused the exception to be raised are printed by the specific NetHandl procedure.

5. Accessing Frames and Subnets

The collection of modules which contain the subnet accessing and maintaining routines are referred to as NetServer routines. There is sometimes some confusion about this term, since the "Net" part of it can refer to both ZOG subnets (or ZOGNets) as well as the physical network of PERQs.

All accesses to frames and subnets in ZOG are accomplished through procedures in module NetHandl. It contains all of the upper level procedures necessary to operate on any frame or subnet within the network of EtherNet connected PERQs, regardless of its physical location(s).

5.1. NetHandl Procedures

The network server handles inter- and intra-machine requests for access to frames and subnets. The location of a frame is neither given to the calling routine, nor is it expected from the calling routine; all that information is maintained and utilized at lower levels.

For convenience, these routines in Module NetHandl may be broken into five categories:

- Frame access Routines
- Frame modification Routines
- Subnet access Routines
- Utility Routines
- NetServer Error Message Routines

5.1.1. Frame access routines

These routines support the ZOG requirments of reading and writing frames. They are:

- RdF - Read Frame
- RdFH - Read Frame Header
- RdFTtl1 - Read Frame Title
- OpnF - Open a frame for modification
- CrFr - Create a frame
- CrF - Create a frame within a specified subnet
- ClrF - Clear a frame

- ClsF - Close and write a frame
- QuitF - Close (do not write) a frame
- ErF - Erase (delete) a frame
- MvF - Move a frame between subnets

5.1.1.1 RdF - Read Frame

This routine takes a frame id and a pointer to a frame record, and loads the record with the contents of that frame. Errors may occur if the frame did not exist, the machine(s) which held that frame were inaccessible, the frame contained bad data, etc.

Up to ten pages of memory (256 words per page) will be passed and parsed by this call.

5.1.1.1.1 Procedure NetHandl.RdF

5.1.1.1.1.1 Verify that the frame pointer <> nil. If not so, exit RdF.

5.1.1.1.1.2 Call Procedure NetString.PrsFid. It parses the frame ID assigning SID(Subnetid) and Fnr(Frame Number).

5.1.1.1.1.3 An Attempt is then made to read the frame by a call to function ZAccess- Procs.ReadFrame.

5.1.1.1.1.4 If the frame is found, then Function NetMakeDel.ClrFP is called to clear out any old contents of the frame, by releasing the pointers to the text.

The pointers to the text are released, but the actual FsTyp nodes containing text are saved in a linked list. This is done to avoid calling the Pascal, New and Dispose routines, procedures with high overhead. It is essentially, ZOG's own garbage collection mechanism.

5.1.1.1.1.5 The Frame Header is then parsed into a record by Procedure BaseLib.ParseFH.

5.1.1.1.1.6 If that is successful, the frame body is then parsed into a record by Procedure NetHandl.ParseF.

5.1.1.1.1.7 Checks the boolean variable, IsAgent (local to the module). If IsAgent is true then ZOG is running an agent and Procedure NetHandl.DelayReturn is called.

Procedure NetHandl.DelayReturn will delay returning control to the calling routine for 1.5 seconds. This is done so that when one machine is running an agent that accesses another machine, that person on the machine being accesed can continue to process in zog. Otherwise, his machine would be flooded with interrupts.

5.1.1.2 RdFH - Read Frame Header

Given a Frame Id and a pointer to a frame header record, this routine loads the record with the contents of that frame's header. Errors may occur if the frame did not exist, the machine(s) which held that frame were inaccessible, or the frame contained bad data, etc.

Read Frame Header differs from Read Frame in that only one 256 word page of memory will be passed back to the caller as opposed to a maximum of ten.

5.1.1.2.1 Reading a frame header (RdFH)

This is the path that control follows when a call is made to a routine in the netserver. RdFH is used because it is fairly typical. To translate to any other access (read) routine, simply change the names of the procedures accordingly.

5.1.1.2.1.1 Verify that the frame header record pointer is not NIL. If it is, return an error.

5.1.1.2.1.2 Call ZAccessProcs.ReadHeader

After initializing the global variables needed to access the network (and returning an error if something was amiss), ReadHeader will attempt to locate the subnet that contains the frame. Then, it will try to read the frame header:

5.1.1.2.1.2.1 If the subnet does not exist, return an error.

5.1.1.2.1.2.2 If the current machine contains the primary copy of the subnet, call NetServ.RdFH -- to obtain the header and exit

5.1.1.2.1.2.2.1 NetServ.RdFH -

- Test to see if the subnet exists on the local machine. If not, there is an inconsistency in the subnet indexes. Return an error.
- Calculate the page number of the first frame. There are 10 pages/frame so this is easy.
- Turn off ethernet interrupts before reading from the disk.

- Read in the page header.
- Turn interrupts back on.
- If the first byte is null, the frame does not exist. Return an error.
- Parse the frame into the frame record.
- If the frame is protected, return an error.
- Return success.

5.1.1.2.1.2.3 If the remote host is listed as being 'up', call ZNet.ZReadHeader to read it from a remote machine and exit if successful,

5.1.1.2.1.2.3.1 ZNet.ZReadHeader

Note: The following algorithm is repeated a maximum of 'MaxRetries' (3) times.

- Recast the ZOG message buffers as ReadHeader message buffers.
- Load up the parameters of the message buffers.
- Call ZOGMsg.SndRcvRecord to send the message over the Ethernet.
- If the returned status was not success, repeat.
- If the message buffer contains an error, repeat.
- Call ZOGMsg.ReceiveBuffer to get the header page.
- Calls Function ZogMsg.ReceiveBuffer to get the frame body.
- If that failed, repeat the initial request.

5.1.1.2.1.2.3.1.1 Recast the ZOG message buffers as ReadHeader message buffers.

5.1.1.2.1.2.3.1.2 Load up the parameters of the message buffers.

5.1.1.2.1.2.3.1.3 Call ZOGMsg.SndRcvRecord to send the message over the Ethernet.

SndRcvRecord does a synchronous send/receive pair between two machines connected by the EtherNet. The routine sends a message across the EtherNet and waits (with a timeout) for a reply. Errors are returned accordingly. (For details see p. 68)

The messages generated by SndRcvRecord cause EtherNet exceptions to be raised on the local (sending) machine and target (receiving) machine, and these then raise the 'E10ReceiveDone' exception, locally and within ZOG. The local E10RecieveDone handler of SndRcvRecord handles the acknowlegement and reply of the target machine to the local machine. The E10RecieveDone handler at the ZOG system (in Module Zog) level handles the receiving of the request of the sending machine and processing it.

5.1.1.2.1.2.3.1.3.1 ZOG.E10ReceiveDone

The EtherNet exception handler in ZOG is invoked when a remote machine sends the current machine a message. This exception handler contains nested handlers to protect ZOG from dying when additional messages are received while ZOG is processing ethernet messages.

5.1.1.2.1.2.3.1.3.1.1 Change the mouse image to the hollow arrow. This is purely cosmetic.

5.1.1.2.1.2.3.1.3.1.2 Call ZOGMsg.HandleMsg to get the message buffers. If an error occurs, ignore the message. The other machine will resend it if it is important enough.

For details f the inner workings of ZOGMsg.HandleMsg see p. 74.

5.1.1.2.1.2.3.1.3.1.3 Pass the message buffers to ZNetServer.ZNetServer for processing.

Module ZNetServer is the counterpart of Module NetHandl on the remote machine. It invokes the local routines which will return the necessary data. It is simply a case statement which uses the input message id to determine which routine should be called. In this example, it will call Procedure ZNetServer.XZReadHeader.

5.1.1.2.1.2.3.1.3.1.3.1 Procedure ZNetServer.XZReadHeader.

XZReadHeader is the equivalent, on the remote machine, to Procedure NetHandl.RdFH on the local machine. Its method is very differnt from that of Procedure NetHandl.RdFH because of the fact that it must perform its task on a remote machine. Notice, however, that both call Procedure NetServ.RdFH_ to do the low level reading of the frame. It does the following: **Recasts variables local to the Procedure as ethernet request and reply types.**

Calls Function ZNetProcs.ZReadHeader to load the header block into the message buffer. ZNetProcs is the counterpart to ZAccessProcs and handles access on a remote machine. For details on the operation of NetServ.RdFH_ see p. 57.

- Verifies that the subnet exists, via a call to Function ZogNetServer.Chk- SnRecord. ChkSnRecord is very similar to GetSnRecord, except that the subnet information passed along with the request is assumed to be correct. This eliminates the need to request it from the MasterNodes Subnet Index. So only the local subnet index needs to be examined to make sure that the information is correct.

- Otherwise call NetServ.RdFH - to load the buffer with the frame header block.

Calls Function ZogMsg.SendRecord to send a reply to the sending machine.. Sends a record to a remote machine and waits for an acknowledgement of receipt of the record. **Sets addresses to be correct, in various records, so that the record can be received on the remote machine.**

Resend Loop. At this point the SendRecord enters a loop to send a request to the other machine, saying, "Well, Go ahead". The loop will attempt to send the request a maximum of NumberResends times (5). To send the request, first the ethernet interrupts are turned off. Next, a call is made to Procedure Ether10IO.E10WIO which starts an EtherNet I/O operation and waits for it to complete. In this case, information is being sent, so E10WIO makes sure the information is sent over the EtherNet. **If an error is detected in sending the message, then exit SendRecord. Otherwise, set the EtherNet Handler State to indicate that the local machine is waiting for the acknowledgement from the remote machine (SWaitAck) and turn on the EtherNet interrupts.**

Got Acknowledgement Time-Controlled Loop. If the acknowledgement is received by the machine sending the message, an interrupt is generated, causing an exception to be raised by the EtherNet MicroCode, thus invoking the local Handler E10ReceiveDone. E10ReceiveDone sees that the EtherNet Handler State indicates that the local machine is waiting for an acknowledgement (SwaitAck), and signals acknowledgement by assigning the EtherNet Handler State to be that of 'Got the Acknowledgement' (SGotAck). If the acknowledgement is received, exit SendRecord. **If after five attempts no acknowledgement is received from the remote machine, then exit SendRecord with an error.**

Calls Function ZogMsg.SendBuffer to send the actual frame header.. If PgCnt = 0 then exit SendBuffer successfully. The buffer is empty, and nothing is sent. Set the Ethernet Handler State to indicate that this machine would like to go ahead and send a buffer (SWaitGo).

Wait for Go Ahead Time-Controlled Loop. At this point a time controlled loop is entered, and its purpose is to wait for an interrupt which indicates that it is all right to send the first buffer. If the interrupt occurs, the local handler E10ReceiveDone is invoked and acknowledgement is sent to the remote machine. If this acknowledgement is sent successfully, the EtherNet Handler State is set to indicate 'Go Ahead and Send the First Buffer' (SSendFirst). Otherwise, the handler is exited, leaving the Ethernet Handler State in the original state. **Set up records with correct addresses to send first buffer.**

Resend Loop.

- Assumes initially that only one page is being sent and puts that page into the buffer to be sent.
- Checks to see if there is more than one page to transfer. If so, sets the buffer page size to two and puts the second page in the buffer.
- Turns the Ethernet interrupts off and sends the Zog Buffer Packet with a call to Procedure Ether10IO.E10WIO. If sent successfully. sets the EtherNet Handler State to be that of 'Waiting for a Buffer-Received Acknowledgement' (SWaitBufAck) and turns Ethernet interrupts on. Otherwise, exits SendBuffer with an error.

Buffer Received Acknowledgement Time-Controlled Loop. At this point, again another time-controlled loop is entered. This time it is waiting for an interrupt indicating that the buffer was sent. If that interrupt occurs, the local handler E10ReceiveDone is invoked. It first examines the acknowledgement from the machine that received the buffer, for correctness. If the acknowledgement is correct and if all the information has been sent, the EtherNet Handler State is set to indicate that all has been sent (SSentAll) and the handler is exited. Otherwise, the handler attempts (only once) to send the next buffer itself, in the same fashion as SendBuffer. **If after five attempts the buffer has not been sent, then exit SendBuffer with an error. Calls Function ZogMsg.SendBuffer to send the actual frame body.**

5.1.1.2.1.2.3.1.3.1.4 Clean up the mouse image and anything else if necessary.

5.1.1.2.1.2.3.1.4 If the returned status was not success, repeat.

5.1.1.2.1.2.3.1.5 If the message buffer contains an error, repeat.

5.1.1.2.1.2.3.1.6 Call ZOGMsg.ReceiveBuffer to get header page

ReceiveBuffer waits for a buffer from a remote machine connected by the EtherNet.

5.1.1.2.1.2.3.1.6.1 Function ZogMsg.ReceiveBuffer.

5.1.1.2.1.2.3.1.6.1.1 If PgCnt = 0 then exit ReceiveBuffer successfully. The buffer is empty, and nothing is sent.

5.1.1.2.1.2.3.1.6.1.2 Sets up several records to send an acknowledgement to the remote machine. This will acknowledge the 'Go Ahead' message sent by the remote machine during its synchronized execution of Function ZogMsg.SendBuffer. In essence, the local machine is saying, 'Go ahead and send me the information I requested, I am ready to receive it'.

5.1.1.2.1.2.3.1.6.1.3 Resend Loop.

At this point ReceiveBuffer enters a loop to send the acknowledgement from the local machine (which originally requested the information) to the remote machine (the machine sending the information), telling it to send the information that was requested (Go Ahead). It will attempt to send the acknowledgement NumberResends times (5). Each time it attempts to send the acknowledgement, it will enter a time-controlled loop to see if the acknowledgement sent was received by the remote machine. If an error occurs in sending the acknowledgement, then control exits ReceiveBuffer with an error. Otherwise, the EtherNet Handler State is set to indicate that we are ready to receive the information (SWaitGoAck). We then enter the Go Ahead Time-controlled loop (mentioned above) to wait for an interrupt to begin receiving.

5.1.1.2.1.2.3.1.6.1.4 Go Ahead Time-Controlled Loop.

After the acknowledgement is sent, the Go Ahead Time-Controlled Loop begins. Here, the machine which requested the information is waiting for an interrupt so that it can begin receiving information. If the interrupt occurs, an exception is raised and the local handler E10ReceiveDone is invoked. Inside the handler, another buffer, acknowledging the receipt of the buffer, is prepared, so that it can be sent when the buffer is received. Finally, the Ethernet Handler State is changed to indicate that the machine will be receiving information (SReceiving).

Note: It is conceivable that the acknowledgement could be sent so quickly that a second interrupt could be generated before entering this loop. This is not likely, but if it does occur, the buffer will have been received before this time-controlled loop, in the handler, and the ethernet handler state will have been updated to indicate that the machine has gotten all the information (SGotAll).

5.1.1.2.1.2.3.1.6.1.5 If the interrupt is not generated, another attempt is made at sending the Go Ahead Acknowledgement. As usual, there are five attempts.

5.1.1.2.1.2.3.1.6.1.6 Wait Receive Time-Contolled Loop.

At this point the receiving machine enters a time-controlled loop to begin waiting for an interrupt indicating that the information requested has arrived. If that interrupt occurs, an exception is raised and the local handler E10ReceiveDone is again invoked, with the Ethernet Handler State being SReceiving Inside the handler, a check is made to see that the information comes from the correct source and that the page numbers are correct. Finally, after all this effort, the requested information is transfered from the sending machine to the receiving machine and the receiving machine makes an attempt to send an acknowledgement to the sending machine. If the acknowledgement is not sent, the handler doesn't worry because the sending machine will time out.

Note : If the information was received in the first Time-Controlled loop, then the Ethernet Handler State will reflect this and ReceiveBuffer will exit successfully.

5.1.1.2.1.2.3.1.6.1.7 If the interrupt was received, then the information requested will have been received in the handler and the EtherNet Handler State will indicate that all information has been received (SGotAll). In this case, ReceveBuffer exits successfully. Otherwise, ReceiveBuffer will timeout.

5.1.1.2.1.2.3.1.7 Calls Function ZogMsg.ReceiveBuffer to get the frame body.

5.1.1.2.1.2.3.1.8 If that failed, repeat the init request.

5.1.1.2.1.2.4 Else, probe the remote machine. If it responds, send the request and read the header and exit if successful.

5.1.1.2.1.2.5 If the current machine has a backup copy, use NetServ.RdFH -,

For details of the operation of NetServ.RdFH_see p. 57.

5.1.1.2.1.2.6 Else, try all of the other backup copies with ZNet.ZReadHeader.

For details on the operation of ZNet.ZReadHeader see p. 58.

5.1.1.2.1.2.7 Otherwise, give up and return an error.

5.1.1.3 RdFTtl1 - Read Frame Title

This routine returns the first line of the title string of a frame, if it can find the frame (see RdF). It performs similar to RdF, and does some very limited frame parsing to find and return the title string.

RdFTlt1 accepts as parameters a frame ID and an array(called Title) for the title to be returned. It again is almost identical to procedure NetHandl.RdF, except that it doesn't parse the entire frame into a frame record structure. Essentially, the procedure will;

5.1.1.3.1 Parse the frame ID.

5.1.1.3.2 Attempt to read the frame.

5.1.1.3.3 If the frame is found, it is read and a check is made to ensure that the first characters represent a frame title (the FIRST occurrence of +T+ in the ZBH file not preceded by other item code characters, denotes a frame title).

5.1.1.3.4 If it is a valid title, RdFTl1 parses the first character string of the frame title into the var character array, Title, to be returned.

5.1.1.3.5 Checks the boolean variable, IsAgent (local to the module). If IsAgent is true then ZOG is running an agent and Procedure NetHandl.DelayReturn is called.

Procedure NetHandl.DelayReturn will delay returning control to the calling routine for 1.5 seconds. This is done so that when one machine is running an agent that accesses another machine, that person on the machine being accesed can continue to process in zog. Otherwise, his machine would be flooded with interrupts.

5.1.1.4 OpnF - Open a frame for modification

This routine reads a frame, and informs the machine upon which that frame resides that it is to be opened for writing. This locks that frame from being written by all other ZOG users and agents. A user (or agent) may only have one frame open at any given point in time.

Errors may occur if the frame can not be found, or if the frame is already open. If the frame is already open to the current user, and a reopen is tried, the frame is closed, and available for subsequent use. This handles the case of a machine aborting during the time it has a frame open on a remote machine.

5.1.1.4.1 Procedure NetHandl.OpnF

5.1.1.4.1.1 Verify that the frame pointer <> nil. If not so, exit OpnF.

5.1.1.4.1.2 Call Procedure NetString.PrsFid to parse the frame ID. Again, it assigns variables SID(Subnet ID) and Fnr(Frame Number).

5.1.1.4.1.3 An attempt is made to open the frame thru a call to Function ZAccessProcs.OpenFrame.

5.1.1.4.1.3.1 Calls Function ZAccessProcs.CheckUser to make sure the user is currently logged in.

5.1.1.4.1.3.2 Calls Function ZogNetServer.GetSnRecord which takes a subnet name and hashes first into the local subnet index. If the subnet is not found there, then it hashes on the subnet index of the master node. A side effect of this function is that the file is opened for reading, via a call to Function NetServ.OpnSn- from Function ZogNetServer.OpnSnRecord from GetSnRecord.

5.1.1.4.1.3.3 Checks to see that the Primary node containing the subnet is up.

5.1.1.4.1.3.4 If the current machine contains the Primary copy of the frame, then a call is made to Function NetServ.OpnF-.

5.1.1.4.1.3.5 Function NetServ.OpnF-.

This function will open a frame for modification,lock it from access by other users, and read the frame from the file.

- EtherNet interuppts are turned back on and the subnet record pointer ids saved for Function NetServ.ClsF- when the frame is closed.
- Calls Procedure BaseLib.ParseFH to parse the frame header into record form for display.
- If the frame is Read or Write protected and the current user is not an owner, then delete this frame from the list of open frames and exit with an error.
- If the frame can be accessed, turn the EtherNet interrupts off, and read body pages via Procedure FileSystem.FsBlkRead.
- Turn EtherNet interrupts back on and save the body page count in the open frame list for Function NetServ.ClsF- for the closing of the frame.
- Calls Function NetServ.GOpnUser to see if the current user has another frame open. If so, exit with an error.
- Calls Function BaseLib.TSidValid to check for a valid Subnet id.
- Calls Function NetServ.GSnRec which obtains the subnet from the local subnet index. The subnet was put there during initalization or during the call to Function NetServ.OpnSn- from Function ZogNetServer. GetSnRecord.

- If no errors have occured to this point, ethernet interrupts are turned off, via Procedure ZEInt.EIntOff. Then the frame header is read into a buffer by a call to Procedure FileSystem.FSBlkRead, and the frame header buffer count is assigned (FHBCnt).

- Calls Function NetServ.GOpnFid to see if the current frame is already open. If so, the ethernet interrupts are turned back on, via a call to ZEInt.EIntOn, and this user cannot access that frame.

- Calls Function NetServ.CrOpnRec which adds another frame to the list of open frames (refered to as locking the frame).

5.1.1.4.1.3.6 Function ZNet.ZOpenFrame

ZOpenFrame is designed to open a frame on a remote machine. It will attempt to send the message three times to the remote machine before it gives up. The number three was chosen arbitrarily. It uses two pointer types, defined in Module ZogMsgDefs, they are OpenF0PTyp and OpenF1PTyp. They represent the request and reply EtherNet packets respectively. For deails on ZogMsg.ReceiveBuffer see p. 61. The routine follows.

5.1.1.4.1.3.6.1 Intializes Variables and increments retries to begin a makeshift loop using labels.

5.1.1.4.1.3.6.2 The variables, local to the module ZNet, OutMsgP and InMsgP, are recast as open frame request and reply pointer types, to be used as such.

5.1.1.4.1.3.6.3 Prepare the EtherNet request packet, through various assignments.

5.1.1.4.1.3.6.4 Function ZogMsg.SndRcvRecord

SndRcvRecord does a synchronous send/receive pair between two machines connected by the ethernet. The routine sends a message across the Ethernet and waits (with a timeout) for a reply. Errors are returned accordingly.

The messages generated by SndRcvRecord cause ethernet exceptions to be raised on the local (sending) machine and target (recieving) machine, and these then raise the 'E10ReceiveDone' exception, locally and within ZOG. The local E10RecieveDone handler of SndRcvRecord handles the acknowlegement and reply of the target machine to the local machine. The E10RecieveDone handler at the ZOG system (in Module Zog) level handles the receiving of the request of the sending machine and processing it.

5.1.1.4.1.3.6.4.1 ZOG.E10ReceiveDone

The EtherNet exception handler in ZOG is invoked when a remote machine sends the current machine a message. This exception handler contains nested handlers to protect ZOG from dying when additional messages are received while ZOG is processing ethernet messages.

5.1.1.4.1.3.6.4.1.1 Change the mouse image to the hollow arrow. This is purely cosmetic.

5.1.1.4.1.3.6.4.1.2 Call ZOGMsg.HandleMsg to get the message buffers. If an error occurs, ignore the message. The other machine will resend it if it is important enough.

For details of the inner workings of ZOGMsg.HandleMsg see p. 74.

5.1.1.4.1.3.6.4.1.3 Function ZNetServer.ZNetServer.

Module ZNetServer is the counterpart of Module NetHandl on the remote machine. It invokes the local routines which will return the necessary data. It is simply a case statement which uses the input message id to determine which routine should be called. In this example, it will call Procedure ZNetServer.XZOpenFrame.

5.1.1.4.1.3.6.4.1.3.1 Procedure ZNetServer.XZOpenFrame.

XZOpenFrame is the equivalent, on the remote machine, to Procedure NetHandl.OpnF on the local machine. Its method is very differnt from that of Procedure NetHandl.OpnF because of the fact that it must perform its task on a remote machine. It is important to notice, that both will call Procedure NetServ.OpnF_ to do the low level reading of the frame.

5.1.1.4.1.3.6.4.1.3.1.1 Recast variables local to the Procedure to be ethernet request and reply types.

5.1.1.4.1.3.6.4.1.3.1.2 Function ZNetProcs.ZOpenFrame.

ZNetProcs is the counterpart to ZAccessProcs and handles access on a remote machine. For details on NetServ.OpnF_ see p. 65.

- Verifies that the subnet exists, via a call to Function ZogNetServer.Chk- SnRecord. ChkSnRecord is very similar to GetSnRecord, except that the subnet information passed along with the request is assumed to be correct. This eliminates the need to request it from the MasterNodes Subnet Index, so only the local subnet index needs examined to make sure that the information is correct. ChkSnRecord also opens the subnet by inserting it into the local subnet index and making sure that the file exists.

- Otherwise call NetServ.OpnF – to open, lock and read the frame.

5.1.1.4.1.3.6.4.1.3.1.3 Calls Function ZogMsg.SendRecord to send a reply to the sending machine.

For details of the inner workings of ZOGMsg.SendRecord see p. 60.

5.1.1.4.1.3.6.4.1.3.1.4 Calls Function ZogMsg.SendBuffer to send the actual frame header.

For details of the inner workings of ZOGMsg.SendBuffer see p. 60.

5.1.1.4.1.3.6.4.1.3.1.5 Calls Function ZogMsg.SendBuffer to send the actual Frame body.

5.1.1.4.1.3.6.4.1.4 Clean up the mouse image and anything else if necessary.

5.1.1.4.1.3.6.4.2 Function ZOGMsg.SndRcvRecord.

5.1.1.4.1.3.6.4.2.1 Establishes a time deadline for receipt of the reply record from the target machine.

5.1.1.4.1.3.6.4.2.2 Sets up several records to be sent over the ethernet, by the local machine. These are recast as ZogMsgPTyp's.

5.1.1.4.1.3.6.4.2.3 Resend Loop

At this point SndRcvRecord enters a loop to send the request for information to the target machine. The loop will attempt to send the request a maximum of NumberResends times (5). To send the request, first the ethernet interrupts are turned off. Next, a call is made to Procedure Ether10IO.E10WIO which starts an EtherNet I/O operation and waits for it to complete. In this case, information is being sent, so E10WIO makes sure the information is sent over the EtherNet.

5.1.1.4.1.3.6.4.2.4 If an error is detected in sending the message, then exit SndRcvRecord. Otherwise, set the EtherNet Handler State to indicate that the local machine is waiting for the acknowledgement from the remote machine and turn on the EtherNet interrupts.

5.1.1.4.1.3.6.4.2.5 Wait Reply Time-Controlled loop.

A time controlled loop is simply a loop that terminates after a certain period of time. This is here so that the local machine can wait for an acknowlegement from the target machine, saying that it has received the request for information.

If the acknowledgement is received by the local machine, an interrupt is generated, causing an exception to be raised by the EtherNet MicroCode, thus invoking the local Handler E10ReceiveDone.

E10ReceiveDone sees that the EtherNet Handler State indicates that the local machine is waiting for an acknowledgement (SwaitSndAck), and signals acknowledgement by assigning the EtherNet Handler State to be that of waiting for a reply (SWaitReply). If the acknowledgement has been received, exit the resend loop. Otherwise, continue to attempt to send the message, up to five times, and exit with an error if an acknowledgement is never received.

5.1.1.4.1.3.6.4.2.6 Got Reply Time-Controlled Loop

When an acknowledgement is received, another time controlled loop is entered, waiting for an interrupt which will again invoke the local handler E10ReceiveDone. This time, the Ethernet Handler State is indicating that the local machine is waiting for a reply (SWaitReply). If the exception is raised, E10ReceiveDone sets up the acknowledgement packet and the packet of information to be recieved. It then assigns the EtherNet Handler State to be that of 'Got the Reply' (SGotReply). At this point, SndRcvRecord will exit successfully.

5.1.1.4.1.3.6.4.2.7 TimeOut.

5.1.1.4.1.3.6.5 Calls Function ZogMsg.RecieveBuffer if the reply came back successfully to receive the frame header.

5.1.1.4.1.3.6.6 Calls Function ZogMsg.ReceiveBuffer to receive the buffer containing the frame body.

5.1.1.4.1.4 If this fails, another attempt to open the frame may be made, due to the side effect of the updating of the local subnet index. This occurs in ZogNetServer.GetSnRecord which is called by ZAccessProcs. OpenFrame.

5.1.1.4.1.5 If the frame is found, Calls Function NetMakeDel.ClrFP to clear out the old contents of the frame.

5.1.1.4.1.6 Calls the parsing routines, BaseLib.ParseFH and NetHandl.ParseF, to parse the frame header and body into their approppriate records.

5.1.1.4.1.7 Checks the boolean variable, IsAgent. If IsAgent is true then ZOG is running an agent and Procedure NetHandl.DelayReturn is called.

5.1.1.5 CrFr - Create a frame

This routine will create a frame, and will open it for modification. Errors will occur if the frame already exists, if the subnet which contains the frame is inaccessible (e.g. machine down), if the subnet to contain the frame doesn't exist, etc. CrFr allows users to create ANY specified frame (subject to the above restrictions), typically through the rdi action.

This routine differs from procedure CrF (q.v.) in that any arbitrary frameid may be specified and the corresponding frame itself created. CrF, by comparison, creates (and opens) the "next" frame in the specified subnet. They also differ in the parameters that the two procedures accept. CrFr accepts a frame Id(Fid) and frame pointer, whereas CrF accepts a subnet ID(Sid) and a frame pointer.

5.1.1.5.1 Procedure NetHandl.CrFr

5.1.1.5.1.1 Checks if the frame pointer(FP) = nil. If so, exit CrFr.

5.1.1.5.1.2 Calls Procedure NetString.PrsFid to parse the frame ID and return the subnet ID(Sid) and frame number(Fnr).

5.1.1.5.1.3 An Attempt is then made to create the next frame thru a call to Function ZAccessProcs.CreateFrame. Again, the side effect of ZogNetServer. GetSnRecord may make it necessary to retry CreateFrame.

5.1.1.5.1.4 Calls Procedure NetHandl.ClrF to clear the body of the frame record without disturbing the frame header.

5.1.1.5.1.5 Calls Procedure BaseLib.ParseFH to parse the frame header buffer(FHBP) into record(FHP).

5.1.1.5.1.6 If successful, this procedure will update statistics to show one more frame has been created (StatsLib.IStatsFCr).

5.1.1.5.1.7 Checks the boolean variable, IsAgent (local to the module). If IsAgent is true then ZOG is running an agent and Procedure NetHandl.DelayReturn is called.

Procedure NetHandl.DelayReturn will delay returning control to the calling routine for 1.5 seconds. This is done so that when one machine is running an agent that accesses another machine, that person on the machine being accesed can continue to process in zog. Otherwise, his machine would be flooded with interrupts.

5.1.1.6 CrF - Same, but create within a subnet

CrF creates and opens for modification the next available frame at the end of a specified subnet. For example if the curient last frame in subnet FOO is FOO91, CrF("FOO") will create frame FOO92. This routine is typically used during the process of Top-Down Frame Creation (TDFC), where the new frames that are linked to the parent are simply created at the "end" of the subnet.

Errors may be caused by the subnet not existing, etc.

By comparison, routine CrFr will create (and open) an arbitrary frame with a specific frame ID. The Procedure itself is identical to CrFr, Except;

- In Parameters Accepted.
- In step 2 of CrFr, instead of calling PrsFid, a call is made to Function BaseLib.TSidValid, which checks that the subnet name is valid(11 or fewer alphabetic characters).
- In step 3 of CrFr, instead of calling CreateFrame, a call is made to Function ZAccessProcs.CreateNextFrame. It returns success if the next frame has been created. Incidently, the next frame is calculated in Function NetServ.CrF - which is called by ZAccessProcs.CreateNextFrame.

5.1.1.7 ClrF - Clear a frame

ClrF will clear the body of a frame record, leaving the header intact. Thus. the invisible parts of a frame (e.g. owner, protection, creation bits) remain unchanged, whereas the visible portion (e.g. text, options, pads, etc.) are cleared. As a side effect, the global pads frame of the cleared frame is set to the ZOG default global pads frame (currently GPads1).

will fail only when a NIL frame record pointer is passed to it; there is NO frame checking in this routine: the calling routine MUST insure that the frame in fact exists!

This procedure is called by CrFr and CrF after they have created the next frame. Its lone parameter is the frame pointer(FP).

5.1.1.7.1 Procedure NetHandl.ClrF.

5.1.1.7.1.1 First checks to make sure that the frame pointer passed to it, (FP) <> nil.

5.1.1.7.1.2 Calls Procedure NetMakeDel.ClrFBody which essentially does the work of calling other procedures to clear(set to nil) the frame body of the FP passed to ClrF.

5.1.1.7.1.3 Procedure NetMakeDel.ClrFBody.

This procedure is responsible for clearing out (setting to NIL) the body of a frame. It will not clear out the frame header. Its lone parameter is the frame pointer that it is to clear, (FP). The procedure itself makes two procedure calls. NetMakeDel.RelFBody (p. 98) realeases the pointers of the various fields of the frame body, and NetMakeDel.IniFBody reinitializes all of the fields of the frame body to NIL (except GPads).

5.1.1.7.1.3.1 Procedure NetMakeDel.IniFBody.

IniFBody takes the frame pointer which has just had its body fields released, and initializes these fields to nil. The only excep- tion to this is the field GPAD, which is assigned the value of DefGPad, a variable local to the module, which was initialized to 'GPads1' during the initialization of zog in Pro[illegible]edure NetMakeDel.IniNetMakeDel.

5.1.1.8 ClsF - Close and write a frame

This routine sends the (modified) frame to its host machine, and instructs the host to overwrite the old version of the frame. It will clear the lock on the opened frame.

It can fail when the host machine cannot be accessed, or the frame was never opened, etc.

Its only parameter is the frame pointer (FP) to be rewritten.

5.1.1.8.1 Procedure NetHandl.ClsF.

5.1.1.8.1.1 If the frame pointer (FP) <> nil, then the frame ID of the fr[illegible] is parsed into a subnet ID and Frame Number by Procedure Netstring. F[illegible]d.

5.1.1.8.1.2 If the frame is valid, upon return from PrsFid, then the ethernet interrupts are turned off by a call to Procedure ZEInt.EIntOff.

Just one point to be made here. The ethernet interrupts are generally turned off when disk I/O is being performed, and then reenabled when the I/O is done. That is not the case in this procedure. It is done just to be safe.

5.1.1.8.1.3 Calls Procedure NetHandl.WrF to write the contents of the frame pointer(FP) to into the frame pointer buffer(FBP)

5.1.1.8.1.4 Calls Procedure ZEInt.EIntOn to turn on ethernet interrupts.

5.1.1.8.1.5 Function ZAccessProcs.CloseFrame

CloseFrame handles the closing of the frame on the primary machine and closing on a secdondary machine with different routines. This will become evident in the description of CloseFrame which follows;

5.1.1.8.1.5.1 Calls Function ZAccesssProcs.CheckUser to make sure the user is currently logged in.

5.1.1.8.1.5.2 Calls Function ZogNetServer.GetSnLocal which hashes into the local subnet index for the correct subnet and returns true if found. As mentioned, it checks only the local subnet index because the frame should have been opened.

5.1.1.8.1.5.3 Checks the local servers table to make sure the primary node is up. If not listed as up, calls Function ZogNetServer.Probe to see if the primary node is actually up. If so, the local servers table is updated. If not, exit with an error.

5.1.1.8.1.5.4 Checks to see if all machines with secondary copies are up in the same manner as described above. If a secondary node is not up then set to false an entry in the array SecUpdate (An array repre- senting the status of of secondary machines).

5.1.1.8.1.5.4.1 If the current machine contains the primary copy of the frame then call Function NetServ.ClsF – to close the frame.

For details on the inner workings of NetServ.ClsF_ see p. 76.

5.1.1.8.1.5.4.2 Otherwise, the primary copy resides on another machine and Function ZNet.ZCloseFrame must be used to write and close the frame on a remote machine.

5.1.1.8.1.5.4.2.1 Function ZNet.ZCloseFrame.

ZCloseFrame is designed to close a frame on a remote machine. It uses types, from Module ZogMsgDefs, ClsF0PTyp as a close frame request packet, and ClsF1PTyp and ClsF2PTyp as close frame reply packets.

5.1.1.8.1.5.4.2.1.1 Initializes the local variables for loop using labels.

5.1.1.8.1.5.4.2.1.2 Recasts (restructures) the outgoing message (outMsg) and ingoing message (inMsg) to frame request and reply packets.

5.1.1.8.1.5.4.2.1.3 Prepares the EtherNet Request packet.

5.1.1.8.1.5.4.2.1.4 Calls function ZogMsg.SndRcvRecord to send a request to close a frame on the remote machine and receive a reply to that request. If t successful, then start all over, step 1.

5.1.1.8.1.5.4.2.1.4.1 Function ZogMsg.SndRcvRecord.

SndRcvRecord does a synchronous send/receive pair between two machines connected by the ethernet . For details on Function ZOGMsg.SndRcvRecord i see p. 68. The routine sends a message across the Ethernet and waits (with a timeout) for a reply. Errors are returned accordingly. The messages generated by SndRcvRecord cause ethernet exceptions to be raised on the local (sending) machine and target (recieving) machine, and these then raise the 'E10ReceiveDone' exception, locally and within ZOG. The local E10RecieveDone handler of SndRcvRecord handles the acknowlegement and reply of the target machine to the local machine. The E10RecieveDone handler at the ZOG system (in Module Zog) level handles the receiving of the request of the sending machine and processing it.

5.1.1.8.1.5.4.2.1.4.1.1 ZOG.E10ReceiveDone on the target machine receives the message.

The EtherNet exception handler in ZOG is invoked when a remote machine sends the current machine a message. This exception handler contains nested handlers to protect ZOG from dying when additional messages are received while ZOG is processing ethernet messages. **Change the mouse image to the hollow arrow. This is purely cosmetic.**

Function ZOGMsg.HandleMsg. This is a boolean function that returns true if there is a valid ZOG request. The message is taken from EtherNet packet form and put into a ZOG message record that must be handled. It returns false for those messages not of the ZOG record protocol. It does the following:

- Check to see if there is a legal ZOG record message.
- If the message received is a request for a probe, and machine names match, then send a probe reply. Thus, the probe is handled right here and HandleMsg returns false.
- If the message is an acknowledgement then return a value of false. These are ignored at this level to avoid infinite loops that can occur with two machines that get out of synchronization and begin sending Ack messages back and forth.
- Otherwise, send an acknowledgement to the sending machine and transfer the received message from the buffer to a ZogMsgPTyp and return true.

Function ZNetServer.ZNetServer..

Module ZNetServer is the counterpart of Module NetHandl on the remote machine. It invokes the local routines which will return the necessary data. It is simply a case statement which uses the input message id to determine which routine should be called. In this example, it will call Procedure ZNetServer.XZCloseFrame.

Procedure ZNetServer.XZCloseFrame.. XZCloseFrame is the equivalent, on the remote machine, to Procedure NetHandl.ClsF on the local machine. Its method is very different from that of Procedure NetHandl.ClsF because of the fact that it must p rform its task on a remote machine. It is important to notice, that both will call Procedure NetServ.ClsF to do the low level writing and closing of the frame.

Recast variables local to the Procedure to be ethernet request and reply types.

Function ZOGMsg.SendRecord. Sends a record to a remote machine and waits for an acknowledgement of receipt of the record. **Sets addresses to be correct, in various records, so that the record can be received on the remote machine.**

Resend Loop. At this point the SendRecord enters a loop to send a request to the other machine, saying, "Well, Go ahead". The loop will attempt to send the request a maximum of NumberResends times (5). To send the request, first the ethernet interrupts are turned off. Next, a call is made to Procedure Ether10IO.E10WIO which starts an EtherNet I/O operation and waits for it to complete. In this case, information is being sent, so E10WIO makes sure the information is sent over the EtherNet. **If an error is detected in sending the message, then exit SendRecord. Otherwise, set the EtherNet Handler State to indicate that the local machine is waiting for the acknowledgement from the remote machine (SWaitAck) and turn on the EtherNet**

interrupts.

Got Acknowledgement Time-Controlled Loop. If the acknowledgement is received by the machine sending the message, an interrupt is generated, causing an exception to be raised by the EtherNet MicroCode, thus invoking the local Handler E10ReceiveDone. E10ReceiveDone sees that the EtherNet Handler State indicates that the local machine is waiting for an acknowledgement (SwaitAck), and signals acknowledgement by assigning the EtherNet Handler State to be that of 'Got the Acknowledgement' (SGotAck). If the acknowledgement is received, exit SendRecord. If after five attempts no acknowledgement is received from the remote machine, then exit SendRecord with an error. Calls Function ZogMsg.ReceiveBuffer to receive the frame body to be written. Again, if unsuccessful, exit erroneously.

Function ZNetProcs.ZCloseFrame.. Calls Function ZogNetServer.GetSnLocal, which checks the local subnet index for the subnet containing the frame. Since ZCloseFrame is called only from the machine containing the primary copy of the frame, this is merely a double check to make sure the frame is indeed there.

Function NetServ.ClsF–. ClsF_ is designed to write and close a frame of the primary copy of a subnet. Calls FunctionNetServ.GOpnUser and assigns the value of the frame on the open record list to a variable local to ClsF–. Calls Procedure NetServ.SetModFH to set modification information of the frame. Modification information includes a new version number, the user modifiying, date, time, and whether modification was performed by an agent. Turn off EtherNet interrupts, via Procedure ZEInt.EIntOff. Calls Proc cedure NetServ.WrFH to write the frame header to a buffer. Turn Eth- erNet interrrupts back on, via Procedure ZEInt.EIntOn. Writes the header page and body pages of the modified frame to a file, via Procedure FileSystem.FSBlkWrite. If the frame gets smaller, due to the modification, a page of zeroes is written to the file to terminate the frame body. If the frame became larger, calls Procedure NetSev. SetFileLen to update the file to the new number of pages in the frame body.

- Turns the EtherNet interrupts back on.
- Calls Procedure NetServ.ErOpnRec to remove the open frame record from the list of open frames, thus unlocking the frame.
- Lastly, prepare modification information to be added to the file Change. Log. The file Change.Log stores information about every frame that is modified.
- Add Modification information to file Change.Log, via Procedure BaseLib. AppStrFile.

AppStrFile will append the string to the end of a file.

Calls Function ZogMsg.SendRecord to send an acknowledgement to the machine sending the frame body, indicating the frame body was received.

Function ZOGMsg.SendBuffer. If PgCnt = 0 then exit SendBuffer successfully. The buffer is empty, and nothing is sent. Set the Ethernet Handler State to indicate that this machine would like to go ahead and send a buffer (SWaitGo).

Wait for Go Ahead Time-Controlled Loop. At this point a time controlled loop is entered, and its purpose is to wait for an interrupt which indicates that it is all right to send the first buffer. If the interrupt occurs, the local handler E10ReceiveDone is invoked and acknowledgement is sent to the remote machine. If this acknowledgement is sent successfully, the EtherNet Handler State is set to indicate 'Go Ahead and Send the First Buffer' (SSendFirst). Otherwise, the handler is exited, leaving the Ethernet Handler State in the original state. **Set up records with correct addresses to send first buffer.**

Resend Loop.

- Assumes initially that only one page is being sent and puts that page into the buffer to be sent.
- Checks to see if there is more than one page to transfer. If so, sets the buffer page size to two and puts the second page in the buffer.
- Turns the Ethernet interrupts off and sends the Zog Buffer Packet with a call to Procedure Ether10IO.E10WIO. If sent successfully, sets the EtherNet Handler State to be that of 'Waiting for a Buffer-Received Acknowledgement' (SWaitBufAck) and turns Ethernet interrupts on. Otherwise, exits SendBuffer with an error.

Buffer Received Acknowledgement Time-Controlled Loop. At this point, again another time-controlled loop is entered. This time it is waiting for an interrupt indicating that the buffer was sent. If that interrupt occurs, the local handler E10ReceiveDone is invoked. It first examines the acknowledgement from the machine that received the buffer, for correctness. If the acknowledgement is correct and if all the information has been sent, the EtherNet Handler State is set to indicate that all has been sent (SSentAll) and the handler is exited. Otherwise, the handler attempts (only once) to send the next buffer itself, in the same fashion as SendBuffer. **If after five attempts the buffer has not been sent, then exit SendBuffer with an error. Clean up the mouse image and anything else if necessary.**

5.1.1.8.1.5.4.2.1.5 If successful, call function ZogMsg.SendBuffer to send the frame body to the remote machine. If not successful, then start all over at step 1.

For details of the inner workings of ZOGMsg.SendBuffer see p. 61.

5.1.1.8.1.5.4.2.1.6 If SendBuffer was successful then calls Function ZogMsg.ReceiveRecord to receive the acknowledgement sent by the machine which received the frame body. If unsuccessful, start all over from step 1.

5.1.1.8.1.5.4.2.1.6.1 Function ZogMsg.ReceiveRecord.

ReceiveRecord receives a record from another machine. It is used only in CloseFrame function of Zog, because there are more acknowledgements that the SndRcvRecord can handle in closing a frame.

5.1.1.8.1.5.4.2.1.6.1.1 Sets the address from where the message should be received.

5.1.1.8.1.5.4.2.1.6.1.2 Sets the EtherNet Handler State to indicate that the sending machine is waiting for a reply (SWaitRcv).

5.1.1.8.1.5.4.2.1.6.1.3 Got Reply Time-Controlled Loop.

At this point the procedure will enter a time controlled loop waiting for an interrupt which will again invoke the local handler E10ReceiveDone. This time, the Ethernet Handler State is indicating that the local machine is waiting for a reply (SWaitRev). If the exception is raised E10ReceiveDone assigns the EtherNet Handler State to be that of 'Got the Reply' (SGotRev). At this point, ReceiveRecord will exit successfully.

5.1.1.8.1.5.4.2.1.6.1.4 If the interrupt is never received, ReceiveRecord will time out.

5.1.1.8.1.5.4.2.1.7 If the acknowledgement is received, calls Function ZogMsg.ReceiveBuffer to obtain the updated frame header information of the frame that was closed.

For details of the inner workings of ZOGMsg.ReceiveBuffer see p. 60.

5.1.1.8.1.5.4.3 Following this, Each secondary copy of the frame is updated. If the current machine has a secondary copy then, calls Function NetServ.Update to write and close the secondary copy.

5.1.1.8.1.5.4.3.1 Function NetServ.Update

UpDate is called only in the event that the current machine has a sec- ondary copy of the subnet of the frame. It is very similar to Function NetServ.ClsF_ with the exception to the following two items;

- In the beginning, It must obtain its subnet information from the local subnet index instead of the list of open frames. This is because the primary copy has been written and closed, and the open frame record has been removed from the list of open records.

- Lastly, this information is not added to the file Change.Log, because it lists only what frames have been modified, not each individual frame and backup copy modified.

5.1.1.8.1.5.4.4 If the secondary copy belongs on a remote machine then calls Function ZNet.ZCloseFrame. Remember, a frame can have copies on as many machines as the creator of the subnet specified.

5.1.1.8.1.5.4.5 If there were no secondary update failures (all secondary machines were up), then CloseFrame was successful and exit.

5.1.1.8.1.5.4.6 Otherwise, calls the nested Procedure ZAccessProcs.CloseFrame.SavSecUp- date. This will store is file sec.update the frame number,subnet ID, version number, date, time, curusername and a list of machine names (server names) of those secondary updates which failed.

5.1.1.8.1.6 Checks the boolean variable, IsAgent(local to the module). If IsAgent is true then ZOG is running an agent and Procedure NetHandl.DelayReturn is called.

5.1.1.9 QuitF - Close (do not write) a frame

This routine clears the write protect lock on an opened frame. It essentially, closes a frame without writing to the subnet file. It can fail when the machine on which the frame resides is inaccessible, or when the frame was not open to begin with.

It accepts as its only parameter, a frame pointer pointing to the frame to be closed.

5.1.1.9.1 Procedure NetHandl.QuitF.

- If the frame pointer (FP) <> nil, then the frame ID of the frame is parsed into a subnet ID and Frame Number by Procedure Netstring. PrsFid.

- Calls Function ZAccessProcs.QuitFrame which takes care of closing the frame and seeing to it that the frame is not written. Quit- frame will only check the local subnet index for for the subnet of the frame to be written. This is because, presumably the frame was opened(by OpnF), in which case, the subnet was added to the local subnet index and there should be no need to look elsewhere.

- Checks the boolean variable, IsAgent(local to the module). If IsAgent is true then ZOG is running an agent and Procedure NetHandl.DelayReturn is called.

5.1.1.10 ErF - Erase (delete) a frame

Erase frame takes a frame id and informs the machine on which that frame resides to delete that frame. The frame may not be opened (and thus write-protected), nor may it be a non-existant frame, nor on an unavailable machine, etc.

Note: If the frame to be erased is the last in a subnet, the subnet will be truncated (e.g. a call to CrF would create that frame). Also, in the same case, if the previous (contiguous) frame(s) had been deleted, the subnet would be truncated to exclude it (them), too.

5.1.1.10.1 Procedure NetHandl.ErF.

- Calls Procedure NetString.PrsFid to parse the frame ID and assign the variables Sid(subnet ID) and Fnr(frame number).

- An attempt is then made to delete the frame thru a call to Function ZAccessProcs.EraseFrame.

- If this fails, another attempt to delete the frame may be made, due to the side effect of the updating of the local subnet index. This occurs in Function ZogNetServer.GetSnRecord which is called by EraseFrame.

- If the Frame has been successfully deleted, a call is then made to Function StatsLib.StatsOn. If StatsOn evaluates to true then statistics are recorded.

5.1.1.11 MvF - Move a frame

This routine takes a frame from one subnet and moves it to a new subnet. It is not currently implemented. If implemented, it will create a new frame in the destination subnet, copy the original into that new frame, close the new frame, and delete the old frame.

5.1.2. Frame modification routines

These routines modify existing frames. In general, if the frames can not be opened (and thus write-locked) the routines will fail. They will a[illegible] .ail if the frames (or their host machines) are not accessible. Frame modification routines are generally called via agents, which explains why these frames are open when these routines are executed. In all three of the these routines the processing occurs in two places, in the open record list and to the local copy of the frame record.

5.1.2.1 AddOwnF - Add an owner to a frame

This routine takes a frameID, and adds an owner name to the list of users who 'own' that frame. Its main parameters are the frame pointer (FP) pointing to the frame to which the new owner will be added, and the new owner (UsrId).

5.1.2.1.1 Procedure NetHandl.AddOwnF.

- Calls Function BaseLib.TUsrIdValid to test for a valid user ID. TUsrIdValid returns true if the user ID is valid.
- Calls Procedure NetString.PrsFid to parse the frame ID assigning Sid(SubnetId) and Fnr(frame number).
- Calls Function ZAccessProcs.AddOwner. This function inserts the new owner in the list of owners of a frame header in the open records list. This header is written to the file after the frame is closed.
- If Function ZAccessProcs.AddOwner returns succcessfully, then Func- tion BaseLib.CrFs15P is called to create, or obtain from the saved list of Fs15PTyp's, a new node to store the name of the new owner. CrFs15P then stores the new owners user ID in the node.
- Calls Procedure BaseLib.InsbFs15i to insert the new owner at the beginning of the list of owners of the local frame record(FP↑. Owners).

5.1.2.2 RemOwnF - Remove an owner from a frame

This routine takes a frame id(FP) and a user name(RemName) and removes that user from the list of users who 'own' that frame. An error is returned if the user was not in the list of users.

It is important to remember that only a user can remove himself as the owner of a frame. Effectively, this means that this procedure allows a user to remove himself as an owner of a frame.

5.1.2.2.1 Procedure NetHandl.RemOwnF.

5.1.2.2.1.1 If FP <> nil, then call Procedure NetString.PrsFid to parse the frame ID assigning Sid(SubnetId) and Fnr(frame number).

5.1.2.2.1.2 Call Function ZAccessProcs.RemoveOwner. This function removes the owner in the list of owners of a frame header in the open records list. This header is written to the file after the frame is closed.

5.1.2.2.1.3 If Function ZAccessProcs.RemoveOwner returns succcessfully, then Func- tion NetString.GFs15P is called to return a pointer to the owner in the list of owners(FPt.Owners) of the local frame.

5.1.2.2.1.4 if Function NetString.GFs15P returns successfully, then calls Procedure BaseLib.DelFs15I which removes the Fs15PTyp from the list of owners of the local frame re rd.

5.1.2.2.1.5 Checks the boolean variable, IsAgent (local to the module). If IsAgent is true then ZOG is running an agent and Procedure NetHandl.DelayReturn is called.

Procedure NetHandl.DelayReturn will delay returning control to the calling routine for 1.5 seconds. This is done so that when one machine is running an agent that accesses another machine, that person on the machine being accesed can continue to process in zog. Otherwise, his machine would be flooded with interrupts.

5.1.2.3 SetProtF - Set a frame's protection bits

This routine changes the protection bits of a specified frame. This Procedure accepts as parameters the frame pointer (FP) of the frame whose Protection is to be changed and the Protection (Prot) to be set. The valid protection codes are as follows;

5.1.2.3.1 (UnProt) Unprotected

5.1.2.3.2 (ModProt) Modification protected

5.1.2.3.3 (WrProt) Write protected

5.1.2.3.4 (RdProt) Read protected

5.1.2.3.5 Procedure NetHandl.SetProtF.

- Calls the Function BaseLib.TProtValid which will test for a valid Protection setting. These valid settings were listed on the previous frame.

- If the Frame Pointer (FP) <> nil then call Procedure NetString.PrsFid to parse the frame ID and assigns the subnet ID (Sid) and the frame number (Fnr).

- If parsed successfully, then calls Function ZAccessProcs.SetFrProtec- tion. SetFrProtection will reassign the protection of the frame header in the open record list. This header is what will be written to the file when the frame is closed.

- If Function ZAccessProcs.SetFrProtection Returns successfully then the protection of the local frame record is reassigned.

- Checks the boolean variable, IsAgent. If true, and it should be in this case, call Procedure NetHandl.DelayReturn.

5.1.3. Subnet access routines

These routines allow the user to create and delete subnets. They are:

5.1.3.1 CrSn - Create Subnet

This routine creates a subnet without secondary (backup) subnets. Its only parameter is a Subnet ID (Sid). The entire Routine is merely a call to Procedure NetHandl.CrSnSec, in which CrSn passes it a parameter indicating that it wants to create only one secondary copy. See CrSnSec for details.

5.1.3.2 CrSnSec - Create a subnet with secondary (backup) subnets

This routine creates a subnet and a number of secondary copies. It will fail when the subnet already exists. It accepts as parameters the subnet ID (Sid) and a count of the number of secondary copies (SecCnt) that are to be made of each frame created in this new subnet. The net result of this procedure is that the MasterNodes subnet.index will have the new subnet added to it. No new frames have been created after this procedure is done executing.

All copies will be placed on the current machine.

5.1.3.2.1 Procedure NetHandl.CrSnSec.

- Calls Function BaseLib.TSidValid to test for a valid Subnet ID. A valid subnet ID consists of 11 or fewer characters.

- Initiailizes PrimeNode to -1 (the Current Node) and array SecNodes to -1 (Default values) to be sent to Function ZAccessProcs. CreateSubnet.

- Calls Function ZAccessProcs.CreateSubnet. CreateSubnet will select the primary node to be the the current node unless it is unlisted in the net.servers database, then the primary node becomes the masternode. For secondary nodes, CreateSubnet assigns values to an array representing those machines to recieve backup copies. This information is obtained during initialization from the file Sec.Default and is stored in a global array, imported from Module ZogNetServer. Lastly, this procedure updates the subnet database of the MasterNode and the file Subnet.Index.

- If successful, this procedure will update statistics to show the creation of a new subnet via a call to Procedure StatsLib.IStatsScr.

5.1.3.3 ClrSn - Clear (delete) a subnet

This routine deletes the contents of a subnet, leaving an empty subnet on the host machine(s). It will fail when the subnet does not exist, or when the machine(s) are not accessible. It accepts as its only parameter the subnet ID of the subnet to be deleted (Sid).

5.1.3.3.1 Procedure NetHandl.ClrSn.

- Calls Function BaseLib.TSidValid to test for a valid Subnet ID. A valid subnet ID consists of 11 or fewer characters.
- Calls Function ZAccessProcs.ClearSubnet. ClearSubnet will only work when both the primary machine and secondary machines are up. It will, via a call to Procedure NetServ.ClrSn – , concatenate the directory, filename and extension of the subnet, locate this file and truncate this file to contain only frame 0 of the subnet.
- If successful, this procedure will update statistics to show the creation of a new subnet via a call to Procedure StatsLib.IStatsSDel.
- Checks the boolean variable, IsAgent. If it is true, which should be the case, then Procedure NetHandl.DelayReturn.

5.1.4. Utility routines

These routines provide information regarding netserver internals.

5.1.4.1 Function TFDef - Tests if a specific frame is defined

Accepts as its only parameter the frame to be tested (Fid). Returns true, not success, if the frame is defined.

5.1.4.1.1 Calls Procedure PerQ – String.Adjust which is used to change the dynamic length of a string.

5.1.4.1.2 Calls Procedure NetString.PrsFid to parse the Frame ID, assigning the subnet ID (Sid) and frame number (Fnr).

5.1.4.1.3 An Attempt is then made to read the frame by a call to Function ZAccessProcs.Readframe.

Function ZAccessProcs.ReadFrame is unusual in that it may have to be called twice inorder to actually read a frame. ReadFrame will read a frame locally, if possible. If not, it scans the local servers database for any primary or secondary nodes that are up and running, and sends a request to that node. If none are listed as up, then the local servers database may not be updated, and probes are sent to each of the primary and secondary nodes to see if any of them are actually up. If so, the local servers database is updated. Another attempt may then be able to read the frame.

5.1.4.1.4 If ReadFrame failed a retry may be successful, so this is attempted.

5.1.4.1.5 If ReadFrame Succeeded, calls Function ZAccessProcs.IsSubnetDefined only to get the correct capitalization of Subnet ID and calls Procedure BaseLib.CvIntStr to convert the frame number to a string. A call is then made to Function PerQ-String.ConCat to concatenate the two strings together and the frame ID (Fid) is reassigned the value of the concatenated strings.

5.1.4.1.6 Checks the boolean variable, IsAgent (local to the module). If IsAgent is true then ZOG is running an agent and Procedure NetHandl.DelayReturn is called.

Procedure NetHandl.DelayReturn will delay returning control to the calling routine for 1.5 seconds. This is done so that when one machine is running an agent that accesses another machine, that person on the machine being accesed can continue to process in zog. Otherwise, his machine would be flooded with interrupts.

5.1.4.2 Function TSnDef - Tests if a specific subnet is defined

Accepts as its only parameter the subnet ID (Sid) being tested. Returns success if the subnet is defined. NOTE: a side result of this function is that the case of the var parameter Sid, is modified to match the case of the actual subnet as listed in file :zognet>Subnet.index.

- Calls Function BaseLib.TSidValid to test for a valid Subnet ID. A valid subnet ID consists of 11 or fewer characters.

- Calls Function ZAccessProcs.IsSubnetDefined which first hashes into the local table of subnet records. If not found there it requests the subnet information from the master node.

- If Function ZAccessProcs.IsSubNetDefined returns success then TSnDef will return success.

5.1.4.3 Procedure GSnLoc - Returns the location(s) of a subnet

The location refers to the node number of the machine on which the subnet is located. GSnLoc accepts as parameters the subnet ID (Sid), the Primary node and secondary nodes of the subnet (Primenode and SecNodes), and a count of the number of secondary nodes (SecCnt). The Function is short and follows;

- Calls ZAccessProcs.IsSubnetDefined which will attempt to find the subnet information in the local table of subnet records. If not found there, it will look in the subnet table of the master node.

- If Function ZAccessProcs.IsSubnetDefined fails, GSnLoc will output an error message.

5.1.4.4 Function GHiSn - Returns the number of the highest frame in a subnet

Accepts the subnet ID (Sid) of the subnet to be examined.

- Calls Function BaseLib.TSidValid to test for a valid subnet ID. A valid subnet ID consists of 11 or fewer characters.

- Calls Function ZAccessProcs.GetHiSubnet which will find the highest number in the subnet. GetHiSubnet, like Function ZAccessProcs. ReadFrame, may have to be called a second time for the same reason. That reason being the updating of the local subnet index.

- Retry Function ZAccessProcs.GetHiSubnet if proper signal was generated.

- If successful in finding the subnet and its highest number, assign to GHiSn.

- Checks boolean variable, IsAgent. If true then calls Procedure NetHandl. DelayReturn.

5.1.4.5 Procedure GNxtSn - Returns the next subnet (alphabetically) within ZOG

Accepts as parameters the present subnet ID (Sid), and returns to the calling procedure, via the Var parameter (NxtSid), the next subnet ID.

- Determines the validity of the present subnet (Sid), by a call to the PerQ Pascal Extension Length, and a call to Function BaseLib.TSidValid. A valid subnet Id consists of 11 or fewer characters, but obviously must consist of at least one character.

- Calls the Function ZAccessProcs.GetNextSubnet. GetNextSubnet will get the next subnet in the master index after the present subnet ID. If the present subnet ID = '', the empty string, then the first subnet in the master index is returned in NxtSid. If the string returned by this function, is the empty string, NxtSid = '', then the present subnet ID was the last in the master nodes subnet index. GetNextSubnet will still return success in this case.

- Assigns a value to Parameter Sig. Sig will return false if the present ID is the last in the master index, meaning that there are no more subnets in the index (NxtSid = '').

5.1.4.6 Procedure GCurNode - Returns integer value of the current machine node.

5.1.4.7 Procedure GCurName - Returns name (string) of the current machine node.

5.1.4.8 Procedure GNodeName - Returns the name of a given network machine node.

5.1.4.9 Procedure GIdUser - Returns the user ID of the current user.

5.1.5. Decoding netserver error messages

Most routines in "NetHandl" have a corresponding routine which will display the error message associated with the last invocation of that routine. These routines are generally called with the frame id used in the previous calls, and output errors in the form:

"Could not get frame <frameid> from machine <remote name>" The error message text are printed via function ZError.ZOGErrorMsg. These routines contained in "NetHandl" are:

WrRdFMsg	Read Frame
WrRdFHMsg	Read Header
WrRdfTtl1Msg	Read Title
WrOpnFMsg	Open Frame
WrCrFMsg	Create Frame
WrCpFMsg	Copy Frame
WrAddOwnMsg	Add an owner
WrRemOwnMsg	Remove owner
WrSetPrFMsg	Frame protection
WrClrFMsg	Clear Frame
WrClsFMsg	Close frame
WrQuitFMsg	Quit Frame
WrErFMsg	Erase (delete) Frame
WrMvFMsg	Move Frame
WrCrSnMsg	Create Subnet
WrClrSnMsg	Clear Subnet
WrTSnDefMsg	Test Subnet Defined
WrTFDefMsg	Test Frame Defined
WrGHiSnMsg	Highest Fr in subnet
WrGNxtSnMsg	Next alpha subnet
WrGTopFidMsg	Get top frame Id
WrRTopFidMsg	Read top frame

5.2. Frame Access Example

Below is an example of control flow for accessing frames and subnets in ZOG. All NetServer Routines are very similar, in terms of flow. NetHandl routines call Module ZAccessProcs routines which call either Module NetServ routines for local access, or Module ZNet routines for remote access. Each example points to a *map* of the flow of control to aid the reader and to help show the interaction of machines via the Ethernet.

5.2.1. Network server flow sample - Reading a frame header (RdFH)

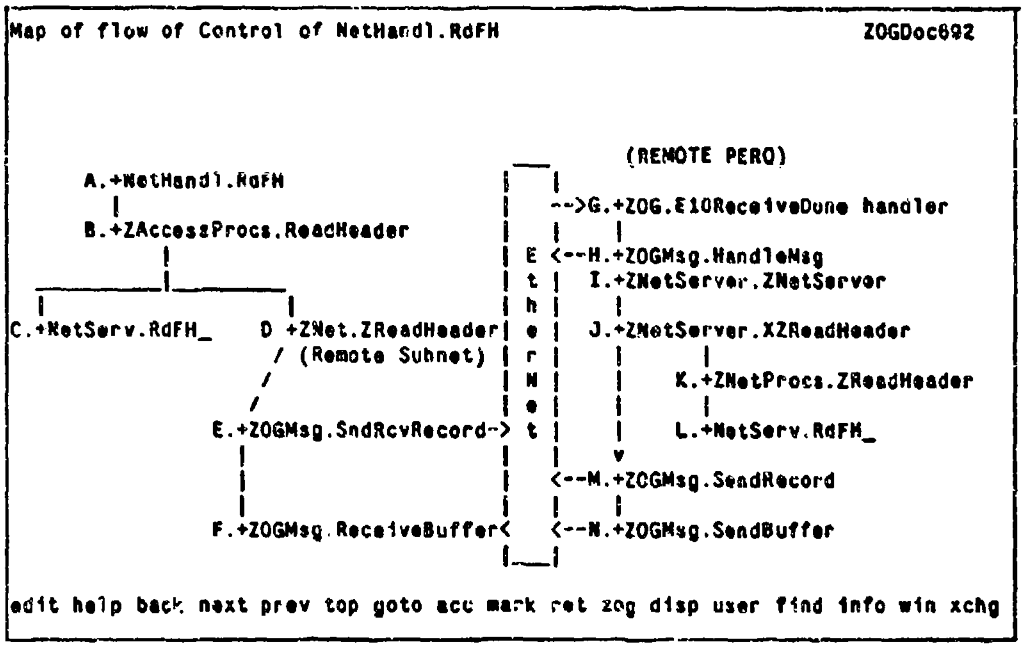
This is the path that control follows when a call is made to a routine in the netserver. RdFH is used because it is fairly typical. To translate to any other access (read) routine, simply change the names of the procedures accordingly. For details on ZAccessProcs.ReadHeader see p. 57.

5.2.1.1 Verify that the frame header record pointer is not NIL. If it is, return an error.

5.2.1.2 Call ZAccessProcs.ReadHeader. This will either load the frame header record, or return an error.

5.2.1.3 Map of flow of Control of NetHandl.RdFH

Ethernet access is done through the NetHandl module, and then down through the NetServer modules. See Figure 5-1 for a pictorial overview of Read Frame Header. A text overview follows.



```
Map of flow of Control of NetHandl.RdFH                               ZOGDoc892


                                                     (REMOTE PERQ)
                                             ___
     A.+NetHandl.RdFH                       |   |
       |                                    |   -->G.+ZOG.E10ReceiveDone handler
     B.+ZAccessProcs.ReadHeader             |   |     |
            |                               | E <--H.+ZOGMsg.HandleMsg
  __________|_________                      | t |   I.+ZNetServer.ZNetServer
  |                  |                      | h |     |
C.+NetServ.RdFH_    D +ZNet.ZReadHeader|    | e |   J.+ZNetServer.XZReadHeader
                   / (Remote Subnet)        | r |     |      |
                  /                         | N |     |    K.+ZNetProcs.ZReadHeader
                 /                          | e |     |      |
             E.+ZOGMsg.SndRcvRecord->       | t |     |    L.+NetServ.RdFH_
               |                            |   |     v
               |                            |   <--M.+ZOGMsg.SendRecord
               |                            |   |     |
             F.+ZOGMsg.ReceiveBuffer<       |   <--N.+ZOGMsg.SendBuffer
                                            |___|

edit help back next prev top goto acc mark ret zog disp user find info win xchg
```

Figure 5-1: Figure

5.2.1.3.1 Reading a frame header (RdFH)

This is the path that control follows when a call is made to a routine in the netserver. RdFH is used because it is fairly typical. To translate to any other access (read) routine, simply change the names of the procedures accordingly. For details on the operation of ZAccessProcs.ReadHeader see p. 57.

- Verify that the frame header record pointer is not NIL. If it is, return an error.

- Call ZAccessProcs.ReadHeader - This will either load the frame header record, or return an error.

5.2.1.3.2 ZAccessProcs.ReadHeader

For details on the inner workings of ZAccessProcs.ReadHeader see p. 5.̇.

5.2.1.3.3 NetServ.RdFH -

- Test to see if the subnet exists on the local machine. If not, there is an inconsistency in the subnet indexes. Return an error.
- Calculate the page number of the first frame. There are 10 pages/frame so this is easy.
- Turn off ethernet interrupts before reading from the disk.
- Read in the page header.
- Turn interrupts back on.
- If the first byte is null, the frame does not exist. Return an error.
- Parse the frame into the frame record.
- If the frame is protected, return an error.
- Return success.

5.2.1.3.4 ZNet.ZReadHeader (Remote Subnet)

For details about the inner workings of ZNet.ZReadHeader see p. 58.

5.2.1.3.5 ZOGMsg.SndRcvRecord

For details of the inner workings of ZOGMsg.SndRcvRecord see p. 58.

5.2.1.3.6 ZOGMsg.ReceiveBuffer

For details of the inner workings of ZOGMsg.ReceiveBuffer see p. 61.

5.2.1.3.7 ZOG.E10ReceiveDone handler

For details of the inner workings of the ZOG.E10ReceiveDone handler see p. 59.

5.2.1.3.8 Function ZOGMsg.HandleMsg

This is a boolean function that returns true if there is a valid ZOG request. The message is taken from EtherNet packet form and put into a ZOG message record that must be handled. It returns false for those messages not of the ZOG record protocol. It does the following:

- Check to see if there is a legal ZOG record message.
- If the message received is a request for a probe, and machine names match, then send a

probe reply. Thus, the probe is handled right here and HandleMsg returns false.

- If the message is an acknowledgement then return a value of false. These are ignored at this level to avoid infinite loops that can occur with two machines that get out of synchronization and begin sending Ack messages back and forth.

- Otherwise, send an acknowledgement to the sending machine and transfer the received message from the buffer to a ZogMsgPTyp and return true.

5.2.1.3.9 Function ZNetServer.ZNetServer

Module ZNetServer is the counterpart of Module NetHandl on the remote machine. It invokes the local routines which will return the necessary data. It is simply a case statement which uses the input message id to determine which routine should be called. In this example, it will call Proceuure ZNetServer.XZReadHeader. For details on the inner workings of ZNetServer.XZReadHeader see p. 59.

5.2.1.3.10 ZNetServer.ZXReadHeader

For details on the inner workings of ZNetServer.ZXReadHeader see p. 59.

5.2.1.3.11 ZNetProcs.ZReadHeader

ZNetProcs is the counterpart to ZAccessProcs and handles access on a remote machine. For details of the operation of NetServ.RdFH_ see p. 57.

- Verifies that the subnet exists, via a call to Function ZogNetServer.Chk- SnRecord. ChkSnRecord is very similar to GetSnRecord, except that the subnet information passed along with the request is assumed to be correct. This eliminates the need to request it from the MasterNodes Subnet Index. So only the local subnet index needs to be examined to make sure that the information is correct.

- Otherwise call NetServ.RdFH_ to load the buffer with the frame header block.

5.2.1.3.12 NetServ.RdFH_

- Test to see if the subnet exists on the local machine. If not, there is an inconsistency in the subnet indexes. Return an error.

- Calculate the page n mber of the first frame. There are 10 pages/frame so this is easy.

- Turn off ethernet interrupts before reading from the disk.

- Read in the page header.

- Turn interrupts back on.

- If the first byte is null, the frame does not exist. Return an error.

- Parse the frame into the frame record.
- If the frame is protected, return an error.
- Return success.

5.2.1.3.13 ZOGMsg.SendRecord

5.2.1.3.14 ZOGMsg.SendBuffer

5.2.1.3.15 Ethernet Communication in RdFH

This diagram, very basically, visually shows the ethernet communication between routines involved in reading a frame header.

```
LOCAL MACHINE                                     REMOTE MACHINE
                 Sends Message (to read FH)
SndRcvRecord ----------------------------------->EIOReceiveDone
                 Sends Acknowledgement                  |
             <-----------------------------------HandleMsg

                 Sends Go Ahead Signal
ReceiveBuffer<-----------------------------------SendRecord
                 Acknowledges the Go Ahead
             ----------------------------------->
                 Sends Frame Header
             <----------------------------------SendBuffer
```

6. ZOG Utilities Modules

In the implementation of ZOG, many utility procedures and functions were written to facilitate the building of the higher-level ZOG procedures. These utility routines have been placed in several modules, which are detailed below:

6.1. BaseLib - Basic miscellaneous utility procedures

The utilites of this module are broken down into the following areas

- General Utilities.
- Frame Header Parsing Utilities.
- Insertion Utilities.
- Initialize and Save Record Utilities.
- Release Memory and Clear Utilities.
- Create and Delete Utilities.
- Get List Entry Utility.

6.1.1. BaseLib - General Utilities

The general utilities are furthur broken down into the following classes.

6.1.1.1 BaseLib - Test Utilities

6.1.1.1.1 Function TIsUc - tests if a char is upper case.

6.1.1.1.2 Function TIsLc - tests if a char is lower case.

6.1.1.1.3 Function TIsAlph - tests if a char is alphabetic.

6.1.1.1.4 Function TIsDig - tests if a char is a digit.

6.1.1.1.5 Function TSidValid - tests if the subnet ID is valid. A valid subnet id consists of 11 or fewer alphabetic characters.

6.1.1.1.6 TProtValid - tests for valid protection setting.

The valid protection types (settings) are constants defined in Module NetDefs. The values specified here are the actual constant values. A ProtTyp, the parameter for this routine, is an integer and is also defined in NetDefs.

- (UnProt) Unprotected

- (ModProt) Modification protected
- (WrProt) Write protected
- (RdProt) Read protected

6.1.1.1.7 Function TUsrIdValid - tests for a valid user ID. The user Id is a string of characters a use enters to log onto the system.

6.1.1.1.8 Function TOwnF - tests if a user is an owner of a frame. It accepts as parameters the frame (FP) and the owner (UsrID) in question.

6.1.1.2 String => Integer, String => Long Integer Conversion Utilities, and Append Utilities

In the conversion routines, going from integer to string, the routines accept a decimal number and use modulo arithmetic to obtain the ASCII equivalent. Going the other way, from integer to string, the routines parse the string and convert it to its decimal equivalent.

Procedure CvIntStr
Convert an integer to a character string.

Procedure CvStrInt
Convert a character string to an integer.

Procedure CvLongStr
Convert a longword integer to a character string.

Procedure CvStrLong
Convert a character string to a longword integer.

Procedure AppStrFile
Appends a string to the end of a file by reading the file into a buffer, appending the string to the buffer, writing and closing the file.

6.1.1.3 BaseLib - Time and Date Utilities.

- Procedure GTimStr : Get Current time of day in hours, minutes and seconds (HH:MM:SS).
- Procedure GTimInt : Get current time in seconds since midnight.
- Procedure CvTimStrInt : Converts a string to time in milliseconds, by parsing a TimStr (HH:MM:SS) and converting first to seconds and then to milliseconds.
- Procedure GDatStr : Get the current days date (DD MM YY).
- Function GDatInt : Get todays date in internal integer format.
- Function CvMonStrInt : Convert month string to its integer form. For example, the month of September has the integer value, 9.

- Function CvDatStrInt : Convert a date string into its integer form. It parses the date string, getting the integer forms of the month, day and year. It then plugs these values into an equation to obtain the integer form of the date string.

6.1.2. BaseLib - Frame Header Parsing Utilities.

- Function ParseLine - parses a line of text from a buffer, in ZBH form, into a string and returns in the Var parameter, Str, only the characters in the line. The routine deletes the three-character ZBH code out of the string returned via Str. ParseLine itself returns a character. The character represents the record stored in the buffer and is the ZBH code for the frame item represented by the input string.

- Procedure ParseFH - parses a frame header buffer, in ZBH form, into a record. It calls ParseLine to parse one line at a time of the buffer into its record form, using the value returned by ParseLine to determine which item is being assigned in the frame header.

6.1.3. BaseLib - Insertion Utilities.

- Procedure InsbFs15l - inserts a string15 type at the beginning of a linked string15 list. One example of its use is when a new owner is added to a frame. Here, a string15 must be added to the record item, FP.owner.

- Procedure InseFs15l - inserts a string15 type onto the end of a linked string15 list.

6.1.4. BaseLib - Initialize and Save Record Utilities.

6.1.4.1 Procedure IniFHP - initializes a frame header record. All record fields are initialized to nil, zero or false, as appropriate.

6.1.4.2 Procedure BaseLib.SavFs15P.

SavFs15P is essentially a garbage collector. It accepts as its only parameter a frame string[15] pointer, (Fs15P) that it is to add to the list of unused Fs15PTyp's. It adds the node sent to it to the front of the list of unused Fs15PTyp's and points the header pointer at this node. In this case, the header pointer is the variable, Fs15PSav, which is local to the module.

6.1.4.3 Procedure SavFHP - saves a frame header record on the list of free FHPTyp's in the same fashion, and for the same reason as SavFs15P.

6.1.5. BaseLib - Release Memory and Clear Utilities

6.1.5.1 Procedure BaseLib.RelFs15P

RelFs15P will release a frame string[15] (Fs15PTyp) pointed to by a frame pointer field. It accepts as its lone parameter a frame string pointer type (Fs15P : Fs15PTyp). This procedure will go thru the list of Fs15PTyp's and call Procedure Baselib.SavFs15P (For details see p. 94), to add a new Fs15PTyp to the front of the linked list of unused Fs15PTyp's. The contents of the Frame string[15] pointer are not altered; it will be cleared out if and when a new frame is created.

6.1.5.2 Procedure RelFH - releases the contents of a frame header record through a call to RelFs15P since the only field needing to be released is the record field, Owners.

6.1.5.3 Procedure ClrFHP - Clears a frame header structure by releasing the frame header via a call to RelFH and reinitializing the header record by calling IniFHP.

6.1.6. BaseLib - Create and Delete Utilities.

- Function CrFs15P - Creates a new string15 pointer type for a frame. This routine either creates the pointer, via the Pascal new statement, or obtains the new node from the saved list of Fs15PTyp's.

- Function CrFHP - Creates a new frame header record. This routine also either creates the nodes or obtains them from the list of free FHPTyp's.

- Procedure DelFs15I - Deletes a string from a string15 list and does not save it on the free list nor does it do a Pascal dispose on it.

6.1.7. BaseLib - Get List Entry Utility.

- Function GEqFs15P - Gets an entry on a string list matching the string passed to the function. This routine does a direct string comparison, testing for the equality of the two strings.

6.2. Fs - String - Routines for the manipulation of Frame String Structures

Fs_String utilities can be broken into the following categories

6.2.1. Fs - String - Length Utility and Write Utilities.

Function Fs - Length

returns the total length of a frame string by going through the linked list of FsPTyp's and calling PERQ - String.Length to obtain the length of each node in the string, adding them together, plus 1 for each (implied) carriage return.

Function Fs - Lines

returns the number of lines in a character string. Each FsPTyp contains one line of text by definition.

Procedure WrFsFile

Writes the entire frame character string to a PASCAL file. The entire write operation is Ethernet interrupt protected.

Procedure WrFsXFile

writes some substring of a frame character string to a Pascal file. The substring must be part of one FsPtyp node.

Procedure WrFs writes the entire frame character string to the video display unit, without using the normal ZOG utility WrStrLn.

6.2.2. Fs – String · Conversion Utilities.

Procedure CvFsStr Converts a frame string to a PERQ Pascal string. Given a linked list of text (FsPTyp), it concatenates the individual strings of the list into one long string of maximum length 255.

Function CvStrFs Converts a PERQ Pascal string to a frame string. Given a string of maximum length 255, it parses that string into a linked list of strings (FsPTyp), where each string in the list is a maximun of 80 characters.

Function SFsP Given a character count into a string, return a frame frame string pointer to point to the line in the corresponding frame string record wherein the character count points.

6.2.3. Fs – String · String Utilities.

- Procedure Fs – Adjust : Changes the length of a frame string to the given value.
- Function Fs – ConCat : Concatenates two frame strings.
- Function Fs – SubStr : Get a sub Portion of a frame string.
- Procedure Fs – Delete : Removes characters to a frame string.
- Procedure Fs – Insert : Insert a string into the middle of a frame string.
- Function Fs – Pos : Finds the position of a string mask in a frame string.
- Function Fs – PosC : Finds the position of a character in a frame string.
- Function Fs – RevPosC : Finds the position of the last occurence of a character in a frame string.
- Function Fs – AppendString : Appends one frame string to another.
- Procedure Fs – CAppend : Appends a character to the end of a frame string.
- Procedure Fs – ConvUpper : Converts a frame string to all upper case.
- Procedure Fs – ConvLower : Converts a frame string to all lower case.

6.3. NetMakeDel · ZOG's internal memory allocation and deallocation routines

The utilites of this module are broken down into the following areas

6.3.1. NetMakeDel - Initialize and Save Record Utilities

6.3.1.1 Procedure IniNetMakeDel - Initializes variables in this module.

6.3.1.2 Procedure NetMakeDel.IniFBody

IniFBody takes the frame pointer which has just had its body fields released, and initializes these fields to nil. The only exception to this is the field GPAD, which is assigned the value of DefGPad, a variable local to the module, which was initialized to 'GPads1' during the initialization of zog in Procedure NetMakeDel.IniNetMakeDel.

6.3.1.3 Procedure IniFP - Initialize entire frame record.

6.3.1.4 Procedure NetMakeDel.SavFsP.

SavFsP is essentially a garbage collector. It accepts as its only parameter a frame string pointer, (FsP) that it is to add to the list of unused FsPTyp's. It adds the node sent to it to the front of the list of unused FsPTyp's and points the header pointer at this node. In this case, the header pointer is the variable, FsPSav, which is local to the module.

6.3.1.5 Procedure NetMakeDel.SavSelP.

SavSelFP is essentially a garbage collector. It accepts as its only parameter a selection pointer, (SelP) that it is to add to the list of unused SelPTyp's. It adds the node sent to it to the front of the list of unused SelPTyp's and points the header pointer at this node. In this case, the header pointer is the variable, SelPSav, which is local to the modul .

6.3.1.6 Procedure SavFP - Save Frame Record in the same fashion as SavSelP and SavFsP, on the local list of saved frame records, FPSav.

6.3.2. NetMakeDel - Release Memory and Clear Utilities

6.3.2.1 Procedure NetMakeDel.RelFsP.

RelFsP will release a frame string (FsPTyp) pointed to by a frame pointer or selection pointer field. It accepts as its lone parameter a frame string pointer [illegible] sP : FsPTyp). This procedure will go thru the list of FsPTyp's and call Procedure NetMakeD[illegible] (see p. 97), to add the new FsPTyP to the front of the linked list of unused FsPTyp's. The co[illegible] the original Frame string pointer are not altered; it will be cleared out if and when a new frame is created.

6.3.2.2 Procedure NetMakeDel.RelSelP.

This procedure is where the actual release of the pointer field of the frame pointer occurs. It accepts as a parameter a selection pointer (SelP : SelPTyp) pointed to by a frame pointer field. It then releases the fields TEXT, ACTION, EXPAND, and EXTRAFLDS of the Selection pointer (SelP), thru a call to procedure NetMakeDel.RelFsP (see p. 97). Lastly, it saves the selection pointer in a linked list, as part of ZOG's own garbage collection mechanism, through NetMakeDel.SavSelP (see p. 97).

6.3.2.3 Procedure NetMakeDel.RelFBody.

The purpose of this procedure is to release the pointers that the fields Title, Text, Options, LPads, Comment, and Accessor point to. Its lone parameter is the overall frame pointer (FP). It releases the pointers by calling the following three procedures for the appropriate fields within the frame body;

6.3.2.3.1 Procedure NetMakeDel.RelSelP.

This procedure is where the actual release of the pointer field of the frame pointer occurs. It accepts as a parameter a selection pointer (SelP : SelPTyp) pointed to by a frame pointer field. It then releases the fields TEXT, ACTION, EXPAND, and EXTRAFLDS of the Selection pointer (SelP), thru a call to procedure NetMakeDel.RelFsP (see p. 97). Lastly, it saves the selection pointer in a linked list, as part of ZOG's own garbage collection mechanism, through NetMakeDel.SavSelP (see p. 97).

6.3.2.3.2 Procedure NetMakeDel.RelFsP, for Frame Comment

6.3.2.3.3 Procedure BaseLib.RelFs15P, for Frame Accessors - NOT USED in PERQ ZOG

6.3.2.4 Procedure RelF - Releases the entire contents of a frame record. This routine merely calls Procedure BaseLib.RelFH to release the frame header and the above procedure, RelFBody to release the frame body.

6.3.2.5 Procedure RelFP - Releases the frame pointer and saves the FPTyp on a list of free FPTyps via NetMakeDel.SavFP.

6.3.2.6 Procedure RelFHP - Releases the frame header pointer and saves the FHPTyp node on a list of free FHPTyps, via NetMakeDel.SavFHP.

6.3.2.7 Procedure ClrFBody.

For details of the inner workings of NetMakeDel.ClrFBody see p. 72.

6.3.2.8 Procedure ClrFP - Clear entire frame record by releasing the frame header and body structures and reinitializing the frame pointer fields.

6.3.3. NetMakeDel - Create and Delete Utilities

The three create routines all act the same in that they obtain the new node either by creating it or getting it from the free list of the particular node type. They then reinitialize the new node.

The two delete routines behave the same in that they remove a node from particular list. Note: No attempt is made to save these deleted nodes in these routines.

Function CrFsP	Creates a new frame string pointer type (FsPTyp).
Function CrFP	Creates a new frame pointer type (FPTyp).
Function CrSelF	Creates a new selection pointer type (SelPTyp) in a frame.
Procedure DelFsl	Deletes a string from a list of frame strings.
Procedure DelSell	Deletes a Selection Pointer type from a list of SelPTyps.
Procedure SetMrkSel	Sets or clears the "continuation mark" in the selection text after the selection character, where the continuation mark is simply a ' ', and the non-continuation is a '-'.

6.4. NetInsert - Routines to manipulate strings into frame data structures

These routines all perform insertions into lists and are broken down into the following areas. An example of the use of NetInsert routines can be seen in Module Fs_String, which calls many of the FsPTyp insertion routines.

6.4.1. NetInsert - Insert at the beginning of a list Utilities

- Procedure InsbFsl - Inserts a (frame) string (string[80]) at the beginning of a list of FsPTyp's.

- Procedure InsbSeli - inserts a selection pointer (SelPTyp) at the beginning of a list of selection pointers.

6.4.2. NetInsert - Insert prior to a node in a list Utilities.

- Procedure InspFs15l - Inserts a new FsPTyp node into a linked list prior to the node currently being pointed to in the list (Fs15P).

- Procedure InspFsl - Inserts a frame string into a linked list prior to the node currently being pointed to in the list (FsP).

- Procedure InspSell - Inserts a selection pointer (SelPTyp) into a list of selection pointers prior to the node currently pointed to in the list (SelP).

6.4.3. NetInsert - Insert after a node in a list Utilities

- Procedure InsaFs15l - Inserts a new FsPTyp node into a linked list after to the node currently being pointed to in the list (Fs15P).

- Procedure InsaFsl - Inserts a frame string into a linked list after to the node currently being pointed to in the list (FsP).

- Procedure InsaSell - Inserts a selection pointer (SelPTyp) into a list of selection pointers after to the node currently pointed to in the list (SelP).

6.4.4. NetInsert - Insert at the end of the list Utilities

- Procedure InseFsl - Inserts a frame string at the end of a list of FsPTyp's

- Procedure InseSell - Inserts a selection pointer (SelPTyp) at the end of a list of selection pointers

6.5. NetOption - Provides utilities for manipulating frame selections

6.5.1. NetOption - Utilities which search for selections based on selection character

- Function GSelF - (PRIVATE) - returns a pointer to the selection in the current list whose selection character matches the specified character, or NIL if no match is found.

- Function GOptF - returns a pointer to an option in the specified frame with the given selection character if it exists, else, the Var boolean, Sig, is set to false. Invokes NetOption.GSelF to do the work.

- Function GPadF - returns a pointer to a local pad in the specified frame with the given selection character if it exists, else, the Var boolean, Sig, is set to false. Invokes NetOption.GSelF to do the work.

6.5.2. NetOption - Utilities which insert selections in the specified frame

- Procedure InsSelF - (PRIVATE) - Generalized selection insertion routine which searches throught the specified selection list (option or local pad) until it finds the appropriate place, based on row and column position for the new selection. It then invokes either NetInsert.InsASelL or NetInsert.InsPSelL to insert the selection in the appropriate slot in the selection list.

- Procedure InsOptF - inserts the specified option into the specified frame's option list via NetOption.InsSelF.

- Procedure InsPadF - inserts the specified local pad into the specified frame's option list via NetOption.InsSelF.

6.5.3. NetOption - Utilities for creating new selections

- Function CvStrSelTxt - returns a pointer to the selection text, in frame string type record, converted from the passed PASCAL string. It does this via a call to Fs - String.CvStrFs to do the actual string parsing into the Frame string record.

- Function CrOptF - Returns a pointer to a new option in the specified frame with the specified selection character at the specified position via calls to NetMakeDel.CrSelF and NetOption.InsOptF.

- Function CrPadF - Returns a pointer to a new local pad in the specified frame with the specified selection character at the specified position via calls to NetMakeDel.CrSelF and NetOption.InsPadF.

- Function GNewOpt - returns the row and column position and selection character for a next new option in the specified frame, based on ASCII sequence of selection characters.

6.6. NetStack - Routines to manipulate the frame stack

Because these routines only stack the frame id's of frames, popping the stack always involves a read of a frame. Thus, they are somewhat inefficient. The routines in Module StackLib overcome this problem by stacking the entire frame record; thus, popping a frame simply involves releasing some records, and modifying some pointers, rather than a correspondingly slower disk read.

6.6.1. NetStack - Initialization Utilities

- Procedure InitFstk - Initializes the fields of a frame buffer stack.

- Procedure InitNetStack - calls Procedure InitFstk to initialize the stack in Module NetStack with the exported frame stack record variable, FPX.

6.6.2. NetStack - General Stack Utilities.

- Procedure RdFstk - Calls Procedure NetHandl.RdF to read a frame from a file into the top of the frame buffer stack.

- Procedure PshFstk - Pushes the Frame Id of the current top frame in the frame buffer stack in preparation for adding the next frame onto the top of the stack.

- Procedure PopFstk - Restores a frame to the top frame of the frame buffer stack by popping the old frameid from the stack, and reading it back into the stack via NetHandl.RdF.

6.6.3. NetStack - Standard Frame Buffer Stack Utilites

These routines are the most commonly used ones, since they use the exported frame stack record, FPX as their default argument.

- Procedure RdFstkX - Reads a frame from a file into the buffer stack X, through a call to

Procedure RdFstk.

- Procedure PshFstkX - Preserves the current top frame in the frame buffer stack X through a call to Procedure PshFstk.

- Procedure PopFstkX - Restores a preserved frame to the top of the frame buffer stack X through a call to Procedure PopFstk.

6.7. NetString - Provides low level string and character manipulation routines

NetString Utilities are broken into the following areas;

6.7.1. NetString - Convert Utilities.

Function CvUcLc converts the passed upper case character to a lower case character. It returns the lower case character.

Function CvLcUc converts the passed lower case character to upper case. It returns the upper case character.

Procedure ConvLower

is used to convert a string of characters to lower case by calling NetString.CvUcLc to do the work.

Function AnyPos returns the position of a string mask in some source text. It converts both the mask and source to lower case, and then invokes PERQ – String.Pos to find the position of the mask (a pattern) in the source string.

Function Narrow converts a double precision integer to a single precision integer by using the Perq Pascal extension intrinsic, shrink.

Function Widen converts a single precision integer to a double precision integer by using the PERQ PASCAL extension intrinsic, stretch.

6.7.2. NetString - String Manipulation.

Procedure Strip this routines removes all leading and trailing carriage returns, line feeds, and spaces from the string, returning only the text of the string.

Function TFsNull Tests if a frame string is null, by examining the linked list of FsPTyp's. The first one encountered whose text record has length greater than zero causes the function to exit returning a value of false. Returns true when the function is passed a nil pointer.

Function GFs15P Will go through a linked list of Fs15PTyps to find the position of the mask (pattern), in the text record of the list elements. Returns a pointer to the node in the list containing that mask.

6.7 ၁. NetString - String Equality Utilities.

Function TEqStrCase

Tests if two string are equal, disregarding case, by calling Procedure Strip to rid of nontext characters. Then it calls PERQ - String.Pos to do the actual comparison, by calling it twice, interchanging the mask and source with each call. If both return the value 1, then the two strings are equal and true is returned.

Function TEqStrSub

Tests if one string is a substring of another string, again calling PERQ - String Pos.

6.7.4. NetString - String < = > Numeric Conversion Utilities

These routines make use of the PERQ PASCAL extensions such as Trunc, Float and Round and Stretch to handle most of the work.

Function RoundLong

Converts a real number to a rounded long integer

Function TruncLong

Converts a real number to a truncated long integer

Function FloatLong

Converts a long integer to a truncated real number

Procedure CvRealStr

Converts a real number to a character string

Function CvStrReal

Converts a character string to a real number

6.7.5. NetString - Time and Date Utilities.

- Procedure CvTimIntStr - Converts the integer time in milliseconds to its character form, by converting the hours, minutes and seconds one at a time and appending a ':' to the returned string before proceding to the next level. The result is a a string in the form HH:MM:SS. This routine is the converse of BaseLib.CvTimIntStr.

- Procedure CvMonIntStr - Converts the integer month to a string date. For example, the integer month, 9, is converted to the string date, 'Sep'. This routine is the converse of BaseLib.CvMonIntStr.

- Procedure CvDatIntStr - Converts a date integer to a date string. It converts the day, month and year to strings, one at a time, then inserts them into the string to be returned. This routine is the converse of BaseLib.CvIntDatStr.

6.7.6. NetString - General Utilites.

- Function TFidValid - Tests if a frame ID is legal by calling Procedure NetString.PrsFid below. Returns true if Fid is valid.

- Procedure PrsFid - Parses the frame ID into Subnet Id and Frame Number parameters to be returned to the calling routine. In the process it checks to see if the subnet ID is valid via BaseLib.TSidValid and if the frame number is less than the maximum allowed in a subnet. If either of the tests fails, it returns the empty string for the subnet ID, zero for the frame number, and sets the Var boolean, Sig, to false.

- Note: Because of this routine's call to BaseLib.TSidValid, and the fact that the Subnet Name is a Var parameter, this routine will return the subnet name with corrected case (i.e., the same case as it appears with in :zognet>Subnet.Index). It is used for precisely this purpose in a number of places throughout ZOG.

6.8. NetLib - Umbrella module for the above routines.

7. Editors

The ZOG system editors provide the interface for users to direct make changes to a frame. ZED is the more general editor used during frame creation as well as to modify existing frames in an arbitrary way. SLED is ZOG's slot editor, which provides a mechanism for controlled filling in of "slots" or blanks on a special "environment" frame.

7.1. The ZOG Editor, ZED

ZED is a general purpose frame editor, with special functions for creating, moving and deleting frame items (Text, Title, Options and Local Pads) as well as accessing "hidden" fields (Frame comment, Item actions, expansion areas and selection next-frames). It can be invoked via Top-Down-Frame-Creation, the "edit" global pad and the ↑De<FrameId> action, each of which eventually call ZEdit.EdFr, the sole entry point into ZED.

The ZED Modules provide all the ZED command parsing and most low-level ZED support routines, and only interface with ZOG via the lower-level subnet handling modules and the screen interface modules.

7.1.1. Module ZEdit

This module provides the entry point from ZOG into ZED, via its procedure EdFr. It also incorporates the ZED Commands parser(s) and implements the commands for editing a frame's "invisible fields" (Comments, actions, nextframes and expansion areas).

7.1.1.1 ZEdit.InitEdit

InitEdit is called to initialize the data structures which will be used throughout the ZOG Editor, ZED. It does the following initializations:

- Create an edit frame record for storing a copy of the frame being edited, via a call to NetMakeDel.CrFP
- Allocate memory for a ZED-wide global character buffer pointed to by BufP (defined and declared in ZEDDefs): Set the buffer's initial length to zero (empty buffer)
- Set the ZED-wide global boolean, NetWalking, to false; (This boolean, EXPORTed from ZEDDefs, means the user has temporarily left the editor, and may want to resume editing in the frame last edited.)
- Set the ZED-wide global boolean, XCHG, EXPORTed from ZEDDefs, to false
- Set the ZED-wide global variable, OldWindow, EXPORTed from ZEDDefs, to 1
- Set the default length (in lines) of a frame to be edited

7.1.1.2 Procedures EdFr and Editing

Procedure EdFr is the entry point into ZED. It performs the initial setup for ZED, then invokes the main ZED Command Parser, Editing.

7.1.1.2.1 ZEdit.EdFr

This routine simply (re)displays the frame to be edited in the current window, invokes the initialization routine, ZEdit.Init, and then invokes the ZED command parser.

- If the user had previously done an "@" or a "h" command, invoke EditAgain to query the user about resuming editing, and do the new frame display
- Otherwise, open the specified frame for editing, and, if necessary, display it in the current window
- Invoke ZEdit.Init to display the ZED messages and titles and reset the global editing variables to an initial state
- Call ZEdit.Editing, the ZED command parser to accept and interpret ZED commands from the user, until global boolean, Exiting, is set to true
- Invoke internal procedure, ZEdit.EdFr.EndZedStat, to reset some of the display functions, and to record ZED ending times for Statistics record keeping

7.1.1.2.2 ZEdit.Editing

This procedure is the top-level command interpreter/processor for ZED. All user commands entered from within ZED are interpreted through a large CASE statement, with the appropriate routine being called to actually perform the ZED command.

7.1.1.2.2.1 Initialize and get command character

- Set the cursor on, place it at the bottom of the screen and reset some global booleans
- Get a character from the keyboard or mouse via ZCanvas.RdKeyEv
- Process the iteration count, if any, and get the command character via ZEdit.DigitProc
- Process the "star" ("*") character, if entered, and get the command character (via ZEdit.StarProc)
- Redisplay the ZED message at the bottom of the window with ZDsplnc.WrtMsg

7.1.1.2.2.2 Begin command parsing loop by checking if a command was entered by the mouse, and if so, translating the mouse button click to a command character via ZEdit.WhichMouse.

7.1.1.2.2.3 Parse the command character entered with a case statement, and invoke the appropriate procedure to execute the intended command.

7.1.1.3 ZED Support Procedures

These are some higher level procedures which implement initialization and exiting code, and as well as some of ZED's higher-level commands.

- ZEdit.Init
- ZEdit.EditAgain
- ZEdit.Wandering
- ZEdit.OutZED
- ZEdit.Quit
- ZEdit.Hlp
- ZEdit.Refresh
- ZEdit.ChgGPad
- ZEdit.RepFrame

7.1.1.3.1 ZEdit.Init

This is the initialization routine called each time ZED starts up. It performs a variety of functions:

- Store old alternate cursor, then clear it; Determine height of window
- Load up Global String, EdMsg, with standard ZED user message
- Display message, "*Edit*" string, and new ZED title in window
- Initialize local and global variables, mostly booleans
- Invoke ZEdUtil.ItemInit to initialize global variables CurSelP and CItem

7.1.1.3.2 ZEdit.EditAgain

This procedure is used when the user has suspending a previous editing session (Module-wide boolean NetWalking is true), and the user is initiating a new session. it queries the user as to which frame he/she wishes to edit, gets the results, and displays the appropriate frame.

- Display query as to whether the user wishes to resume editing the previous frame in the previous window, and obtain reply
- If so, obtain that frame and move to the appropriate window, if necessary

- Otherwise, if there were changes made to the previously edited frame, (XCHG is true) then ask user if he/she wishes to preserve those changes. If so, then write out the frame stored in EdFP.

- Finally, ask the user if he/she wishes to edit the current frame. If so, then open that frame for editing and read it into EdFP, and set the var parameter fid to be the current frame. Else, just clear the messages and redisplay the current frame if it has been changed.

7.1.1.3.3 ZEdit.Wandering

This procedure implements the "@" command, to allow the user to (temporarily) leave the editing session on the current frame, and go "wandering" through the subnets. It sets up global booleans to allow for the return to the previously edited frame via ZEdit.EditAgain.

- Display the appropriate message in the user display line

- Save the changes made with ZEdUtil.Save – Chg (which sets Global XCHG to true)

- Remove the ZED messages and "ZED global pads" via ZEdit.Clean

- Set Module-wide boolean, NetWalking, to true, so EditAgain will be invoked upon reentry into ZED

- Make other window the current window via ZWind.XChange

- Set Global boolean, Exiting, to true, so as to leave EdFr

7.1.1.3.4 ZEdit.OutZED

OutZED implements the Exit ("e") command. It is the routine which causes the edited copy of the frame to be written out to the subnet on the disk.

- Set Module-wide Boolean, NetWalking, to false

- Clean off the ZED related messages and strings on the frame being edited via ZEdit.Clean

- Invoke ZEdUtil.Save – Chg

- If the current user is "GUEST" and changes were made to the frame being edited, give the user a message stating that no permanent changes are being made, and exit

- If changes were made to the frame being edited, then close the frame, via NetHandl.ClsF, and read the new frame into the window structure with ZWind.RdFWind

- Set Global Boolean, Exiting, to true, so as to exit EdFr

7.1.1.3.5 ZEdit.Quit

Quit is used to implement the "q" command. The changes made to the frame are discarded, and the original frame is redisplayed.

- Set Module-wide boolean, NetWalking, to false
- Invoke ZedUtil.Save – Chg
- Erase the ZED messages and "global pads" with ZEdit.Clean
- Do a quit for the currently open frame via NetHand.QuitF
- If changes were made during the editing session, reread and redisplay the original unchanged frame via ZWind.RdFWind and ZWind.DspWind

7.1.1.3.6 ZEdit.Hlp

This routine is very similar to procedure Wandering; it saves the changes that were made, and sets the NetWalking state. It then switches to the other window and displays the top ZED help frame.

- Display the "wandering" message in the user display line
- Invoke ZedUtil.Save – Chg
- Erase the ZED strings on the frame with ZEdit.Clean
- Set Module-wide boolean, NetWalking to true, so ZED can reenter editing of the current frame
- Set Global boolean, Exiting, to true, so as to exit EdFr
- Switch windows with ZWind.XChange
- Read in and display the help frame "Help4" using ZAAction.GoToFrame

7.1.1.3.7 ZEdit.Refresh

This procedure implements the Control-L command; it clears, then redisplays the window containing the frame being edited.

- Save the current cursor position
- Clear the window with ZDisplay.Clear
- Redisplay the frame being edited with ZedUtil.DspFPED
- Display the ZED heading and default message via ZedUtil.DspEditorHdAndTail
- Redisplay the cursor at its old position by recovering the stored old position, then placing the cursor there with ZDspInc.DspTxtPos

- Turn the cursor on with ZCanvas.CursorOn

7.1.1.3.8 ZEdit.ChgGPad

This procedure implements the "G" command, to change the frame's global pad frame.

- Prompt the user for new Global Pad Frame, displaying the current one as a default
- Obtain the user's response via ZedUtil.ReadStr
- If a new Global Pad frame has been specified by the user or GPads1, the ZOG default Global Pad frame, enter it in EdFP↑.gpad
- Redisplay the ZED message in the User display line of the window with ZedUtil.WrtMsg

7.1.1.3.9 ZEdit.RepFrame

This procedure implements the "." (Replace Frame) command. It copies the contents of the specified frame into ZED's frame structure, EdFP.

- Display a prompt for the user to enter the frame id of the frame to copy into the current frame
- Get the user's response via ZedUtil.ReadStr
- If the user's frame id is a valid one, clear the screen, copy the specified frame into EdFP, display the new frame via ZedUtil.DspFPED, initialize with ZEdit.Init, and set the global boolean, XCHG to true, indicating that a change has been made in the editing session

7.1.1.4 Local Utilities

- ZEdit.CmpSel
- ZEdit.DigitProc
- ZEdit.StarProc
- ZEdit.FSelF
- ZEdit.WhichItem
- ZEdit.Mouse
- ZEdit.WhichMouse
- ZEdit.Clean

7.1.1.4.1 ZEdit.CmpSel

This procedure is a low-level routine to match a row and column position in the current window to a selection, returning a pointer to that selection, if a match occurs.

- Starting with the selection pointer passed to this routine, check to see if that selection's row and column position match
- If so, assign the var parameter, NSel, to the comparison selection pointer, set the var boolean found to true, and exit
- If not, get the selection pointed to by the next field, and if it is not NIL, then repeat

7.1.1.4.2 ZEdit.DigitProc

This procedure is used to obtain the [optional] iteration count from the user for his/her commands. It is called from the command interpreters when the first character the user has input is a digit. It returns the iteration count via the global variable, Iteration, and returns the first non-digit character in global variable, Chrr.

- Initialize the local integer variable, int, to 0
- Set the Global variable, ChgIteration to true, to indicate a non-ur 'ty iteration count
- Add the first digit into int and read another character via ZCanvas.RdKeyEv
- Continue adding in (properly for decimal numbers) and reading characters until the first non-digit is entered, at which point assign Iteration to the current value of int, and return

7.1.1.4.3 ZEdit.StarProc

This procedure is used to obtain the command from the user after he/she has entered a "star" ("*"), signifying the command is to apply to the end of the item. It is called from the command interpreters when the first character the user has input is an asterisk. It set the global boolean, Star, and returns the inext character entered in global variable, Chrr.

- Set the Global variable, Star, to true, to indicate that processing is to occur to the end of the current item
- Read the next character into Chrr via ZCanvas.RdKeyEv

7.1.1.4.4 ZEdit.FSelF

This procedure checks to see if the mouse position, passed to it via parameters row and column, falls within the selection box of any of the passed selection type.

- Initialize var boolean, found, to false
- For every non-NIL value of ASel, the passed selection pointer, see if row is within the l0, l1

pair of the selection, and if column is within the c0, c1 pair of the selection

- If so, set found to true, and exit
- Else, get the next selection pointer, ASel↑.nextsel and repeat the checking

7.1.1.4.5 ZEdit.WhichItem

Given a row and column position of the mouse in the frame being edited, this procedure returns a pointer to the selection containing that row, column position, if it exists.

- Set local selection pointer variable, ASel, to the first Option on the frame, and check with FSelF. If successful, just exit
- Set local selection pointer variable, ASel, to the first Local Pad on the frame, and check with FSelF. If successful, just exit
- Set local selection pointer variable, ASel, to the frame text, and check positions locally. If successful, set boolean found to true and exit
- Set local selection pointer variable, ASel, to the frame title, and check positions locally. If successful, set boolean found to true and exit

7.1.1.4.6 ZEdit.Mouse

This procedure implements the mouse selection feature in ZED. If the user clicks the mouse inside any item, this routine will find that item, and put the cursor at the position selected by the mouse.

- Get the row and column of the position of the mouse, and save the current item information in OSelP
- Invoke WhichItem to determine the item inside of which the mouse was clicked
- If no match, give error message and exit
- Else, calculate the new relative position within the selected item
- Update the current item information to reflect the possibly new item and cursor position
- Display the cursor at the new position with ZDspInc.DspTxtPos

7.1.1.4.7 ZEdit.WhichMouse

This procedure is used to interpret a mouse button click as either a "ZED global pad" selection or as a mouse button click for some other purpose, probably cursor movement within the frame.

- If the mouse was clicked somewhere in the bottom row, then call procedure ZedUtil.CheckCommand to interpret the mouse's position as a ZED command from the ZED "global pads" displayed on the bottom row (LenChFrame); this procedure set Global variable, Chrr, to be the first letter of the selected command from the "global pads" list.

- Else, just return Global variable, Chrr, as mousebutton, defined in ZedDefs as Chr(1) (ASCII Control-A).

7.1.1.4.8 ZEdit.Clean

This procedure is used to remove the ZED messages from the display of the edited frame in the current window, prior to leaving ZED.

- Set the character display function to Replace mode via ZCanvas.SetChFunc
- Turn the cursor off, then position it at 1,59 in the frame, the position of the "*Edit*" string
- Display 6 spaces with ZCanvas.PutCh to "erase" that string
- Place the cursor at the user display line with ZDspInc.DspPos, then clear that line with ZDisplay.ClrLine
- Obtain and display the frame's global pads, via ZDisplay.DGPads
- Call ZCanvas.ChangeCanvas to redisplay the whole window

7.1.1.5 Procedures for editing invisible fields

- ZEdit.SavOld
- ZEdit.RestOld
- ZEdit.SmallMouse
- ZEdit.CmdProc
- ZEdit.AltNxtFm
- ZEdit.AltExpAct
- ZEdit.CrExpAct
- ZEdit.RelStr
- ZEdit.AltSelAction
- ZEdit.AltSelExpand
- ZEdit.AltFmComment
- ZEdit.AltFmAction
- ZEdit.AltFmExpansion

7.1.1.5.1 ZEdit.SavOld

This procedure is called whenever any of the "invisible" fields are to be edited. It saves the current selection pointer in the edited frame, and clears the window:

- Save the current selection pointer in global variable, SaveSelP
- Create a new one for use while editing the invisible field, via a call to NetMadeDel.CrSelF
- Call ZedUtil.ItemInit to initialize the new selection pointer and current item structure
- Clear the current window via ZDisplay.Clear
- Make the character display mode the XOR function via ZCanvas.SetChFunc

7.1.1.5.2 ZEdit.RestOld

This procedure is called when the user is finished editing an "invisible" field of a frame. It refills the window with the original frame as it was being edited.

- Set var parameter AStrP to point to the correct string within the current item. Set XCHG to true if there were some changes made in the invisible field
- Release the old current selection pointer, CurSelP
- Redisplay the frame with ZedUtil.DspFPED
- Redisplay the ZED messages with ZedUtil.DspEditorHdAndTail
- Restore the current selection pointer, CurSelP, with SaveSelP
- Initialize the current item structure with ZedUtil.ItemInit

7.1.1.5.3 ZEdit.SmallMouse

This procedure implements mouse selection of characters while in editing "invisible fields" mode of ZED.

- Exit if there is no text (FirstStr = NIL)
- If the mouse's position is within the exisiting text of the field, the calculate a relative position within the field, and display the cursor there, with ZDspInc.DspTxtPos
- Else display an error message via ZedUtil.WrtMsg, beep and set global boolean ChgMsg to true, so that the default ZED message will be displayed on the next pass

7.1.1.5.4 ZEdit.CmdProc

This procedure is the command parser for editing "invisible fields". As such, it only implements a subset of ZED commands, specifically only those dealing with text insertion, replacement/modification and deletion. The other more frame oriented commands are not available.

Note: the user leaves this mode via an escape (INS) key, or an "e" or "q". The latter two also end up exiting ZED, via an Exit(Editing) statement.

- Read in a character from keyboard or mouse, via ZCanvas.RdKeyEv
- Enter the command processing loop: initialize some booleans, display the ZED Default message if appropriate
- If the character entered was a digit, get the rest of the iteration count and the command character
- If the character entered was an asterisk, set boolean Star to true, and get the command character
- Turn the cursor off with ZCanvas.CursorOff
- Do a case statement on the command character, Chrr
- Turn the cursor on via ZCanvas.CursorOn
- Read the next character, and repeat

7.1.1.5.5 ZEdit.AltNxtFm

This is the procedure which allows the user to enter a next frame id for a selection. This is all handled in the user command line.

- If the current item is neither an option or a local pad, beep and exit
- Prepare the message to the user, depending on whether or not there is a pre-existing next frame id for the selection
- Display the message via ZedUtil.WrtMsg and read in the user's reply via ZedUtil.ReadStr, then redisplay the cursor at its previous location via ZDspInc.DspTxtPos
- If the converted (via ZActUtils.CvFid) user-input string is a valid frameid, then install it in the selection's next frame field. Else display an error message, and exit.

7.1.1.5.6 ZEdit.AltExpAct

This is the generic procedure for editing invisible fields (Expansion areas and Actions). The specific procedures prepare a message for display, then call this procedure to do the actual editing.

- Invoke ZEdit.SavOld to save the current state of the edited frame
- Display the header passed to it as a parameter, and the ZED headings
- Display any current text in the field via ZedUtil.ReDsp
- Position the cursor at the beginning of the text via ZDspInc.DspTxtPos
- Turn the cursor on
- Invoke ZEdit.CmdProc to parse the commands for the invisible fields
- When done, invoke ZEdit.RestOld to replace the original frame in the window;
- Turn off the cursor

7.1.1.5.7 ZEdit.CrExpAct

This is a short utility routine used to allocate a Frame String variable for use while editing invisible fields. If, after the user has finished the editing of the invisible field, there is no text in the string variable, then the variable is deallocated, via ZEdit.RelStr.

- Make the local string variable, str, a null string of zero length
- Create the string record via NetHandl.CrFsp, with the null string. The var parameter, astrp, points to the newly allocated string record.

7.1.1.5.8 ZEdit.RelStr

This procedure releases the memory allocated for a string variable that was used while editing invisible fields. It is only called if there was no text at all in the string record when the invisible field editing was terminated.

- Release the allocated memory for the var string pointer via a call to NetHandl.RelFsp
- Make the var string pointer variable, strp, point to NIL

7.1.1.5.9 ZEdit.AltSelAction

This is the "generic" procedure for entering action strings for frame titles (Exit Actions), frame text (Frame Actions) and selections. It is invoked with the "a" command from within ZED.

- Call ZedUtil.Save - Chg
- Create a string record for the action string of the current selection pointer, global

variable, curselp, if one doesn't already exist

- Create the message to be displayed by ZEdit.AltExpAct, depending on the type of action being edited (Exit, Frame or Selection, depending on the type of Item the current selection pointer refers to

- Call ZEdit.AltExpAct to perform the actual editing

- If there is no text in the action string after the editing, release the memory allocated to the string record

- Finally, if the item was a selection, add or remove the "-" accordingly

7.1.1.5.10 ZEdit.AltSelExpand

This procedure implements the "y" command, to edit a selection's expansion string. It does most of its work via ZEdit.AltExpAct.

- Check to see if the current item is the frame's title or text. If so, Beep and exit.

- Invoke ZEdUtil.Save - Chg

- Create a string record via ZEdit.CrExpAct if there is currently no string for the selection's expansion area

- Initialize the string for ZEdit.AltExpAct to display

- Invoke ZEdit.AltExpAct to perform the editing

- Release the memory allocated for the string record if there is no text in the area after editing via ZEdit.RelStr

7.1.1.5.11 ZEdit.AltFmComment

This procedure implements the "C" command, to enter a frame comment.

- Invoke ZedUtil.Save - Chg

- Invoke ZedUtil.Frame - Text to force current item to be the frame's text

- Allocate memory for a frame string record if there is currently none for the Frame Comment

- Initialize the display string, then invoke ZEdit.AltExpAct to do the actual editing functions

- If there is no text in the comment after editing, then release the associated memory via ZEdit.RelStr

7.1.1.5.12 ZEdit.AltFmAction

This procedure implements the "A" command, to enter a frame action.

- Invoke ZedUtil.Save – Chg
- Invoke ZedUtil.Frame – Text to force current item to be the frame's text
- Allocate memory for a frame string record if there is currently none there for the Frame Action
- Initialize the display string, then invoke ZEdit.AltExpAct to do the actual editing functions
- If there is no text in the action string after editing, then release the associated memory via ZEdit.RelStr

7.1.1.5.13 ZEdit.AltFmExpansion

This procedure implements the "Y" command, to enter text into a frame's expansion area.

- Invoke ZedUtil.Save – Chg
- Invoke ZedUtil.Frame – Text to force current item to be the frame's text
- Allocate memory for a frame string record if there is currently no text in the Frame Expansion Area
- Initialize the display string, then invoke ZEdit.AltExpAct to do the actual editing functions
- If there is no text in the expansion area after editing, then release the associated memory via ZEdit.RelStr

7.1.2. ZED Command Implementing Modules

There are two modules within the ZED "library" which contain the bulk of the procedures which actually implement the ZED commands. These are divided by "Whole Frame" commands (Upper Case command letters) and "within Item" commands (Lower Case command letters). These commands are implemented in the following two ZED Modules:

- Module ZEdFram - Whole Frame Editing Commands
- Module ZEdItem - Within Item Editing Commands

7.1.2.1 Module ZEdFram - Whole Frame Editing Commands

This module implements those commands which affect the entire frame, commands such as adding, modifying and deleting items on the frame, moving items around on a frame, or moving from item to item.

7.1.2.1.1 Procedures for moving around in a frame

These procedures allow the user to navigate forward and backwards between items in the frame being edited, so as to get to a specific item.

- ZedFram.GoToText - exported
- ZedFram.GoToPads
- ZedFram.HeadOpns
- ZedFram.TailOpns
- ZedFram.TailPads
- ZedFram.LineFeed - exported
- ZedFram.Escape - exported

7.1.2.1.1.1 ZEdFram.GoToText - exported

This procedure simply makes the current item record point to the frame text.

- Set the global item typed variable, TypeItem, exported from ZEdDefs, to IText
- Make the current selection pointer, CurSelP, point to the edited frame's text (EdFP↑.Text)

7.1.2.1.1.2 ZEdFram.GoToPads

This procedure simply makes the current item record point to the first local pad on the frame.

- Set the global item typed variable, TypeItem, exported from ZEdDefs, to IPads
- Make the current selection pointer, CurSelP, point to the edited frame's first local pad (EdFP↑.LPads)

7.1.2.1.1.3 ZEdFram.HeadOpns

This procedure simply makes the current item record point to the first option on the frame.

- Set the global item typed variable, TypeItem, exported from ZEdDefs, to IOptions;
- Make the current selection pointer, CurSelP, point to the edited frame's first option (EdFP↑.Option).

7.1.2.1.1.4 ZEdFram.TailOpns

This procedure simply makes the current item record point to the last option on the frame.

- Set the global item typed variable, TypeItem, EXPORTed from ZEdDefs, to IOptions
- Make the current selection pointer, CurSelP, point to the edited frame's first option (EdFP↑.Options)

- Continue making CurSelP point to CurSelP↑.NextSel, until the next selection pointer (the new CurSelP↑.NextSel) is nil. This has the effect of stepping through the options until the last one is reached.

7.1.2.1.1.5 ZEdFram.TailPads

This procedure simply makes the current item record point to the last local pad on the frame.

- Set the global item typed variable, TypeItem, exported from ZEdDefs, to IPads

- Make the current selection pointer, CurSelP, point to the edited frame's first local pad (EdFP↑.LPads)

- Continue making CurSelP point to CurSelP↑.NextSel, until the next selection pointer (the new CurSelP↑.NextSel) is nil. This has the effect of stepping through the local pads until the last one is reached.

7.1.2.1.1.6 ZEdFram.LineFeed - exported

This procedure implements the LineFeed (<LF>) command, which steps the user (i.e., the current position on the edited frame) "down" through the frame's items, starting at the current item, and ending at the last local pad. The sequence of stepping through is : Title => Text => Options => Local Pads.

7.1.2.1.1.6.1 If currently at last local pad in frame, BEEP and exit

7.1.2.1.1.6.2 Save any changes made in the current item via ZedUtil.Save - Chg

7.1.2.1.1.6.3 Prepare to enter loop for the number of iteration counts contained in global variable, iteration (default = 1), and for local boolean, EndFrame, signifying we've reached the "bottom" of the frame, being false:

7.1.2.1.1.6.3.1 Do a case for the type of the current item:

TypeItem = IPads Set the Current Selection pointer, CurSelP, to the point to the next local pad, unless there isn't one, in which case, set local boolean, EndFrame to true.

TypeItem = IOptions
Set the Current Selection pointer, CurSelP, to point to the next option. If there isn't one, then set it to point to the first local pad. If there are no local pads, ask the user if he/she wishes to create one. If so, then create one with ZedFram.CrLPad at a default position; if not, set local boolean, EndFrame, to True.

TypeItem = IText Set the Current Selection pointer, CurSelP, to point to the first option, if there is one. If not, then make it point to the first local pad, if there is one. If not, signify by setting local boolean, EndFrame, to true.

TypeItem = ITitle Set the Current Selection pointer, CurSelP, to point to the frame text, even if there is no actual text there.

7.1.2.1.1.6.3.2 Increment the iteration counter, I

7.1.2.1.1.6.4 Place the cursor at the beginning of the new item, and initialize pointers with ZedUtil.ItemInit

7.1.2.1.1.7 ZEdFram.Escape - exported

This procedure implements the Escape (<INS>) command, which steps the user (i.e., the current position on the edited frame) "up" through the frame's items, starting at the current item, and ending at the frame's title. The sequence of stepping through is : Local Pads => Options => Text => Title, just the reverse of ZedFram.LineFeed.

7.1.2.1.1.7.1 If currently at frame title, BEEP and exit

7.1.2.1.1.7.2 Save any changes made in the current item via ZedUtil.Save-Chg

7.1.2.1.1.7.3 Prepare to enter loop for the number of iteration counts contained in global variable, iteration (default = 1), and for local boolean, HdFrame, signifying we've reached the "top" of the frame, being false:

7.1.2.1.1.7.3.1 Do a case for the type of the current item:

TypeItem = ITitle	Simply set local boolean, HdFrame, to True
TypeItem = IText	Set the Current Selection pointer, CurSelP, to point to the frame's title, and set TypeItem accordingly
TypeItem = IOptions	Set the Current Selection pointer, CurSelP, to the previous option. If there isn't one, then set it to point to the frame text, via ZedFram.GoToText.
TypeItem = IPads	Set the Current Selection pointer, CurSelP, to point to the previous local pad, if there is one. If not, then make it point to the last option, via ZedFram.TailOpns, if there is one. If not, set it to point to the frame text, via ZedFram.GoToText.

7.1.2.1.1.7.3.2 Increment the iteration counter, I

7.1.2.1.1.7.4 Place the cursor at the beginning of the new item, and initialize pointers with ZedUtil.ItemInit.

7.1.2.1.2 Procedures to add, modify or delete items in a frame

- ZedFram.PreProc
- ZedFram.GetSelBody
- ZedFram.InitSel

- ZedFram.CrProc
- ZedFram.CrOption
- ZedFram.CrLPad
- ZedFram.MoveDown
- ZedFram.MakeRoom
- ZedFram.GetRow
- ZedFram.ComputeRow
- ZedFram.InTleTxt - exported
- ZedFram.InsOpLp - exported
- ZedFram.Reset+Title+or+Text
- ZedFram.InsDBuf
- ZedFram.DelSel - exported
- ZedFram.XOpns - exported
- ZedFram.RepSel - exported
- ZedFram.CvOpnPad - exported

7.1.2.1.2.1 ZEdFram.PreProc

This is the low-level procedure to create new selections; it is called from several of the selection creation routines. Given a row and column position, this routine creates a string pointer and a selection pointer for the user.

- With a null character and an empty string, allocate memory for a selection string via NetMakeDel.CrFsP
- Create a new frame selection record pointed to by global current selection pointer variable, CurSelP, via NetMakeDel.CrSelF
- Initialize the remaining items in the selection record, and position the cursor at the beginning of the selection via ZedUtil.ItemInit
- If the user is using the "X" (Extend Options) Command, then simply set the global boolean, Xtend to false
- Else, use routine ZEdUtil.Positioning to allow user to move the new selection around via the normal positioning mechanism

7.1.2.1.2.2 ZEdFram.GetSelBody

This procedure is used to obtain the selection text from the user, once the selection itself has been created.

- Call ZedUtil.Insertion to allow entry of characters into current selection record
- Set global boolean, Changed, to true to indicate that a change has been made to the frame
- Use ZEdUtil.Save – Chg to update the selection box variables, and to make the current selection text pointer point to the "first" string in the new selection
- Call ZEdUtil.ItemInit to initialize some of the global current item variables, and to copy the rest of the new strings into the record structure
- Call ZEdUtil.Start – Item to position the cursor at the beginning of the new selection

7.1.2.1.2.3 ZEdFram.InitSel

This procedure is called whenever new selections are being created to automatically supply the first three characters of the first selection string, the appropriate selection character, the period and the dash.

- Use PERQ – String.Adjust to set the dynamic length of the current item's text to 3
- If there are no previous selections, set the new selection character (the first character of the string) to "1"
- Else, if the new selection is not in the same column as the previous selection and the previous selection character is not a digit, then the new selection character is set to "1"
- Else, the new selection character is set to the next character in the (ASCII) sequence from the previous selection character
- Assign the "." and "-", and display the new text at the initial position, an position the cursor immediately after these three characters

7.1.2.1.2.4 ZEdFram.CrProc

This routine handles the top-level selection creation process, after the selection been created and initialized.

- Get the text for the new selection via ZEdFram.GetSelBody
- If there was no text at all entered, then delete the new selection via ZedFram.DelSel, display the default ZOG message and exit
- If there were only three (or fewer) characters (presumably, the selection character, the "." and the "-", then also delete the selection, without inserted the deleted characters in

the delete buffer), display the default ZOG message, and exit

- Call ZEdUtil.MarkSel to make sure that the "·" is the third character of the text if there is no next frame and no selection action
- Call ZEdUtil.UpdSelCh to make the new selection's selection character the first character in the selection text

7.1.2.1.2.5 ZEdFram.CrOption

This is the top level procedure for creating new options in the frame being edited.

- Set Global variable, TypeItem, to IOptions
- Call ZEdFram.PreProc to create the new option and new string record
- Use NetOption.InsOptF to insert the new option in the correct place within the frame record pointed to by EdFP
- Invoke ZEdFram.InitSel to set up the first three characters of the text
- Display the "Enter OPTION TEXT" message
- Invoke ZEdFram.CrProc to finish up the selection creation process
- Display the default ZED message

7.1.2.1.2.6 ZEdFram.CrLPad

This is the top level procedure for creating new local pads in the frame being edited.

- Set Global variable, TypeItem, to IPads
- Call ZEdFram.PreProc to create the new local pad and new string record
- Use NetOption.insPadF to insert the new local pad in the correct place within the frame record pointed to by EdFP
- Display the "Enter PAD TEXT" message
- Invoke ZEdFram.CrProc to finish up the selection creation process
- Display the default ZED message

7.1.2.1.2.7 ZEdFram.MoveDown

This procedure is used to make a "hole" in a group of selections; given a starting selection pointer and a "top" selection pointer, it will move each selection in that range "down" one line.

Starting with the first selection pointer, until it reaches the top selection pointer...

- Set global current selection pointer, CurSelP, to pointer to that selection
- Redisplay that selection with ZDisplay.DspFsP
- Increment all the row variables within the selection by one
- Redisplay that selection, now one line downward, with ZDisplay.DspFsP
- Set the local selection to point to the previous selection, and repeat

7.1.2.1.2.8 ZEdFram.MakeRoom

This procedure "makes room" between two selections for inserting a new selection by figuring out where to insert a new selection the user wishes. In some cases, it will actually make a "hole" in the selections to fit the new one in. This routine is only called when the list of selections is single- spaced so that figuring out where to put the new selection is a non-trivial task.

- Search through the "next" selections until the last one in the current selection's column is found
- If one more than the bottom line of this last selection is "too low" in in the frame to fit (i.e., greater than the length of the frame minus two) then set the var integer, Row, to be this value as a signal to the calling routine (InsOpLp)
- Else, set global boolean, XCHG, to true to signal a change has been made, set row equal to one more than the current selection's bottom line, and call ZEdFram.MoveDown to move the remaining selections down one line to make room for the new one

7.1.2.1.2.9 ZEdFram.GetRow

This routine is invoked during the "I" command to figure out where to "squeeze" in a new selection. It selects a row value either one or two below the "bottom line" of the current selection, depending on how close the next selection (or bottom of the frame) is to the current selection.

- If the "bottom line" of the current selection is greater than the top line of the next selection, set var parameter, Row, to one less than "top" (which had been set equal to two beyond the "bottom line" in the calling routine (ComputeRow))
- Else, set var parameter, Row, equal to the "top" limit

7.1.2.1.2.10 ZedFram.ComputeRow

This routine is the top-level routine for deciding which row to put a new selection in when the "I" command is issued from within the frame's selections.

- Set local variable top to 2 lines "below" the bottom of the current selection
- If there is no next selection, use ZEdFram.GetRow, to get a value for row, either one or two lines below the current selection

- Else if the next selection is in the same column as the current selection, then if the next selection is one line below the current selection, invoke ZEdFram.MakeRoom to either figure out where to place the new selection, or make room for it in the current list of selections

- Else use ZEdFram.GetRow to obtain the row for the new option, since there already is room there for at least one line

- Else, since there are no more selections in the same column, just use ZEdFram.GetRow, with a bottom limit of the bottom of the frame.

7.1.2.1.2.11 ZEdFram.InTleTxt - exported

This procedure implements the "I" command, to insert a new selection when the user is currently in the frame text or frame title.

7.1.2.1.2.11.1 Query the user as to whether he/she wants an option or pad; obtain the response, and if neither is chosen, then display the ZED message, position the user at the frame text via ZedFram.GoToText and ZedUtil.ItemInit, and exit

7.1.2.1.2.11.2 If the user specified a new option, figure out where to place the new option and create it with ZEdFram.CrOption

- If there are already some options, then set the new row two lines below the last option, and the column equal to the column of the current selection

- Else, if there's frame text, set the column equal to 3, and the row to two below the bottom of the frame text

- Else set the row to 5, and the column to 3

- If the row somehow got to be less than 5, then set it to 5, which is the default highest line in which to place an option

- If the row computed above is too low (i.e., below two lines above the bottom of the frame) then change the new column to column 50, and the new row to two above the frame bottom.

7.1.2.1.2.11.3 If the user specified a new local pad, figure out where to place it and and the new local pad with ZEdFram.CrLPad

- Set the new column position to be 64, the default

- If there are local pads, search to the last one. If there's room, then set the row to two below the last local pad - if not, then just one below it, then reset the column to be the same column as the last local pad

- Else, just set the row for the new local pad to be two above the bottom c' the frame

7.1.2.1.2.12 ZEdFram.InsOpLp - exported

This procedure implement the "I" command when the current user position is within the frame's selections.

- If the user is, in fact, in the frame title or text, invoke InTleTxt to implement the command, and exit
- Set the column value for the new selection to be the column of the current selection
- Invoke ZEdFram.ComputeRow (and its daughter procedures) to properly calculate the row for the new selection
- If the current selection is an option, then the new one will also be an option; if the calculated row is too low, then move the column over to column 50, and make the row two above the bottom line; create the new option via ZEdFram.CrOption
- If the current selection is a local, then so will the new one; if the calculated row is too low, then move the column to three to the left of the current selection, and the row to three up from the frame bottom; create the new one with ZEdFram.CrLPad.

7.1.2.1.2.13 ZEdFram.Reset – Title – or – Text

This procedure is used to implement the "D" (delete item) command if the user is currently in the frame title or text. It is called from ZedFram.DelSel.

- Set global variable, XCHG, to true, to signal that the frame has been modified
- Release the global first string record, pointed to by global pointer, FirstStr, via NetMakeDel.RelFsp
- Release the current selection's text, via NetMakeDel.RelFsP and set the text pointer to NIL
- If the current item is the frame title, set the top line to line 1 of the frame
- Set the current (deleted) item's bottom line equal to it's bottom line
- Make its left column position equal to its right
- Init the global variables via ZEdUtil.ItemInit

7.1.2.1.2.14 ZEdFram.InsDBuf

This procedure implements the use of ZED's delete buffer during whole item deletion. The delete buffer is merely a large array of characters. It is used to store characters that were deleted for possible subsequent re-insertion.

Invoke ZedUtil.Start_Item to position the cursor at the beginning of the item

For each line of text in the current item:

- If we've reached the end of the line, then insert a carriage return in the buffer with ZedUtil.InsDelBuf, point to the next line of text in the item, start at (relative) column 1, and increment the row by one

- Else, simply insert the character at the current position into the delete buffer wiht ZedUtil.InsDelB..f, and increment the column position by one

- Replace the selection character with a null via ZEdUtil.UpdSelCh

- Invoke ZedUtil.Start - Item to position the cursor at the beginning of the item

7.1.2.1.2.15 ZEdFram.DelSel - EXPORTed

The top-level procedure for the "D" (delete item) command.

- Set Global boolean, XCHG, to true, to signal frame modification

- If deleted text is to be stored, invoke ZEdFram.InsDBuf

- If the current item is the frame's title or text, then delete the item via ZEdFram.Reset - Title - or - Text, redisplay the cursor at the initial position of the deleted item with ZDspInc.DspPos and exit

- If the current item is an option, then reposition the current selection pointer at the previous item, invoke NetMakeDel.DelSelL to delete the option, release the selection record via NetMakeDel.RelSelP, initialize the previous item via ZEdUtil.ItemInit and Start - Item, and exit

- If tl 'ent item is a local pad, then reposition the current selection pointer to be the previous item. Delete the pad with NetMakeDel.DelSelL, release the associated record with NetMakeDel.RelFsp, initialize the previous item with ZEdUtil.ItemInit and Start - Item

7.1.2.1.2.16 ZEdFram.XOpns - exported

This procedure implements the "X" (eXtend option list) command.

- Initialize, and set up default spacing between new options, based on iteration count

- If there are no options on the frame at all, then figure out the appropriate starting position after the frame text and create the option there via ZEdFram.CrOption

- Else, go to the last existing option via ZEdFram.TailOpns, figure out the correct next position and create the option there via ZEdFram.CrOption

- Go into a loop, doing essentially the same thing as above, adding options at the appropriate spacing after the option before, using ZEdFram.CrOption. This portion of the code has to continually check where the last created option was, so as to recalculate an appropriate next row for the next option.

7.1.2.1.2.17 ZedFram.RepSel - EXPORTed

This procedure implements the "R" (replace entire current selection) command.

- Save the entire text of the option in the delete buffer via ZEdFram.InsDBuf
- Initialize by releasing the old selection's string record, and creating a new one, via NetMakeDel.RelFsP and CrFsp with a null string
- Display the "replace selection text" message, and obtain user's first character input
- If the character is an escape (INS), then just delete the selection via ZEdFram.DelSel, and exit
- Else, use the character as the first (selection) character of the replaced option, display the character, set up the current position variables, move the cursor to the right with ZDspInc.DspCurR and obtain the rest of the selection text with ZEdUtil.Insertion.
- Display the default ZED message with ZEdUtil.WrtMsg

7.1.2.1.2.18 ZEdFram.CvOpnPad - exported

This procedure implements the "V" command. It changes the current option to a local pad or vice versa.

- If the current selection is neither an option or local, Beep and exit
- Invoke ZEdUtil.Save-Chg, set global boolean, ChgMsg, to true so that the default ZED message is displayed later, and obtain the current selection's selection character
- If the current selection is an option, remove the selection from the frame's option list via NetMakeDel.DelSelL and insert it into the local pads list via NetInsert.InsPadF. Change its item type to lPads, and display a confirming message
- If the current selection is a local, remove the selection from the frame's local pad list via NetMakeDel.DelSelL and insert it into the option list via NetInsert.InsPadF. Change its item type to lOptions, and display a confirming message

7.1.2.1.3 Procedures to find selection by selection character

- ZEdFram.CmpSelChr
- ZEdFram.FSelChr
- ZEdFram.FindSel - exported

7.1.2.1.3.1 ZEdFram.CmpSelChr

A lower-level routine to search through a list of selections, attempting to match each one's selection character with the character passed to it. A non-obvious, but utilized side effect of this routine is that it leaves global current selection pointer, CurSelP, pointing to either the matched selection or to the last selection on the list.

- Set booleans to false to initiate the search
- Starting with the current selection (CurSelP↑), check the each selection's selection character with the passed character parameter
- If they match, set var boolean, Found, to true to exit the loop
- Else, if there's a next selection, point to it for the next pass, else, set terminating boolean, EndGroup, to true to exit the loop.

7.1.2.1.3.2 ZEdFram.FSelChr

This is the middle-level selection character search routine.

- Store the current selection pointer, CurSelP, in a local variable
- If there are any options in the frame, set CurSelP to point to the top of the option list. Invoke ZedFram.CmpSelChr to try to find a match. If it was successful, set Global variable TypeItem to be IOptions, and exit.
- If there are any local pads in the frame, set CurSelP to point to the top of the local pads list. Invoke ZedFram.CmpSelChr to try to find a match. If it was successful, set Global variable TypeItem to be IPads, and exit.
- If none of the above searches were successful, Beep and restore CurSelP to point to its previous selection.

7.1.2.1.3.3 ZEdFram.FindSel - exported

This routine implements the "S" (Find selection) command.

- If there are no options or local pads on the frame, beep and exit
- Save the current selection pointer and initialize
- Display the prompt for the selection character to match, and obtain it from the user
- Invoke ZEdFram.FSelChr to find the matching selection
- If the current selection pointer as set by the lower level routines is the same as the saved one, the search was unsuccesful: display the cursor at the start of the old item, display the default ZED message and exit.
- Save any changes that were made on the old current selection, set CurSelP to point to

the new "found" selection, initialize the global variables for the new selection and display the default ZED message.

7.1.2.1.4 Procedures to move items around within a frame

- ZEdFram.AddTotal
- ZEdFram.GlobalExam
- ZEdFram Where
- ZEdFram.Reformat - exported

7.1.2.1.4.1 ZEdFram.AddTotal

This is a support routine for the reformat command. It is used to count up the number of lines in the text of the passed frame string, and add these to a var total.

- For each string (i.e., line) in the passed string list, increment the var counter, sum, and point to the next string in the list.

7.1.2.1.4.2 ZEdFram.GlobalExam

Another support routine for the reformat command. This function returns true if there is enough room in the frame to reposition (reformat) the options.

- Initialize GlobalExam to false, and total (lines) to zero
- If there's a frame title, add the number of lines in the title to total, and one more for the space after the title, and if total is more than the number of lines available, then exit
- If there's frame text, add the number of lines in the text to total, and one more for the space after the title, store the result in local variable oldtotal and if total is more than the number of lines available in the frame, then exit
- If there are no options in the frame, just exit with a true condition
- Else, count the total number of lines in the options (plus spaces between them) that are in the same column as the first option. If the total is too large, exit; otherwise set GlobalExam to true.

7.1.2.1.4.3 ZEdFram.Where

A third support routine for the reformat command. If there are options on the frame in more than one column, this routine asks the user how he/she wishes them reformatted. This procedure returns the new row for the top option in additional columns to re-appear in after the reformat.

- Determine if the two columns are "close" to each other
- Create and display the message about whether the top option should remain stationary

and obtain his/her response (y/n)

- If it is to be moved, set the row to be row 2 in the frame
- Else, set the new row to the option's old row, unless the two columns are too close to each other - if so, then set the new row to be two below the previous options row

7.1.2.1.4.4 ZEdFram.Reformat - exported

This (lengthy) procedure implements the "F" (reformat frame) command.

7.1.2.1.4.4.1 If there are no items on the frame, just exit

7.1.2.1.4.4.2 Initialize and use iteration count to get the spacing to use (LineSpace)

7.1.2.1.4.4.3 Check to see if there is enough room to reformat via ZedFram.GlobalExam; if not, display appropriate message and exit

7.1.2.1.4.4.4 If there is a frame title, and it is not at the top left corner of the frame, move it there, and update its selection box via ZEdUtil.UpdateBox

7.1.2.1.4.4.5 If there is frame text, and it is not in its normal position at 3,1 in the frame, move it there, and update its selection box via ZEdUtil.UpdateBox, and adjust local variable row to reflect where the text is now

7.1.2.1.4.4.6 If there are options, set up the first option's new position, and figure out the correct selection characters to use in the reformat

7.1.2.1.4.4.7 Loop through, reformatting the remaining options.

- If the next option is in a different column, ask the user how to reformat that next column via ZEdFram.Where
- Else place the option LineSpace (1 or 2) lines below the bottom of the last option
- Obtain the number of lines in the current option
- If the current option is at or below the user display line, then force it to be one line above the user display line
- Set the top line of the option (l0) to be the option's row
- Adjust the bottom line of the option (l1) accordingly
- Adjust the option's selection character to be the next character in the ASCII sequence
- Repeat this process for the next option, if it exist

7.1.2.1.4.4.8 Clear the window via ZDisplay.Clear, display the reformatted frame with ZEdUtil.DspFPEd, and the ZED messages via ZEdUtil.DspEditorHdAndTail, and initialize with the cursor where it was before, with ZEdUtil.Iteminit and Start-Item.

7.1.2.2 Module ZEdItem - Within Item Editing Commands

This module implements most of the ZED commands which only affect text within a specific item on the frame. These command mostly comprise inserting, deleting and modifying characters, and cursor movement. Within-item justification is also included, as are a number of miscellaneous routines.

7.1.2.2.1 Procedures for moving around the item

- ZEdItem.MoveCursor - exported
- ZEdItem.ToBlank
- ZEdItem.ToNonBlank
- ZEdItem.FwdWord
- ZEdItem.ForwardW - exported
- ZEdItem.BkWord
- ZEdItem.BackW - exported

7.1.2.2.1.1 ZEdItem.MoveCursor - exported

This procedure handles keyboard-controlled movement within the item, the " " and "<BackSpace>" commands.

- If there is no text within the item, then Beep and exit
- For the iteration count (contained in global variable, iteration), do the following until reach the "end" of the item's text
- If global boolean, BackSpace, is true, then move the cursor backwards one space, via ZDspInc.DspCurL
- Else move the cursor forward one column, via ZDspInc.DspCurR
- Increment the location iteration counter, i
- If the loop has gone beyond the iteration count or past the end of the item, then beep the user

7.1.2.2.1.2 ZEdItem.ToBlank

This is a support procedure for moving forward one word. It searches the current string for the next occurrence of a blank (or <CR>).

- Initialize and save current item information
- Search forward (or backward, depending on the value of Global boolean, Back) through the string, starting at the current position, by utilizing ZDspInc.DspCurR (L), and examining the character at the new position. If a match is found, or the new position is after the end of the current string, then set var parameter, Success, to true.
- If the search was unsuccessful, restore the saved current item information and redisplay the cursor at the original position via ZDspInc.DspTxtPos

7.1.2.2.1.3 ZEdItem.ToNonBlank

Another support routine for moving forward (backward) one word. This procedure is the converse of ZEdItem.ToBlank; it searches the current string from the current position for the first non-blank character.

- Initialize and store the current item information
- Search through the string, starting at the current position, utilizing ZDspInc.DspCurR (L if searching backwards, as indicated by global boolean, Back); if not at the end of the current line of text, and the new current character is not a blank, set var parameter, Success, to true, indicating a successful search.
- If the search was unsuccessful, restore the saved current item information, and redisplay the cursor at its original position via ZDspInc.DspTxtPos

7.1.2.2.1.4 ZEdItem.FwdWord

This procedure does the actual forward movement by word in the current item.

- Check to see if there is a "next" character in the current text. If not, then just Beep and exit.
- Return the cursor back to its original position (ZDspInc.DspCurL) and set global boolean, Back, to false
- If the current position is not at the end of the current line, then if the current character is a blank, go to the next non-blank via ZEdItem.ToNonBlank; else call ZEdItem.ToBlank to move to the next blank, then to the next non-blank, via ZEdItem.ToNonBlank.
- Else, just go to the next non-blank, via ZEdItem.ToNonBlank

7.1.2.2.1.5 ZEdItem.ForwardW - Exported

This procedure implements the ";" (move forward one "word") command, where a word is defined to end at the next space (or <CR>) character.

- If there's no text in the current item, then just beep and exit
- Enter a loop for global variable, Iteration, counts, to call ZEdItem.FwdWord for each word to move forward, incrementing the local iteration counter for each successful movement
- If the number of iterations accomplished is less than the number the user specified, then beep

7.1.2.2.1.6 ZEdItem.BkWord

This procedure does the actual backward movement by word in the current item.

- Check to see if there is a "previous" character in the current text. If not, then just Beep and exit
- Set global boolean, Back, to true to indicate backward searching
- If the current position is past the end of the current text, then go to the first non-blank via ZEdItem.ToNonBlank. If successful, go back to the previous blank, then if successful, forward to the next non-blank.
- Else, if the current character is a blank then go back to the previous non-blank, back to the previous blank, then forward to the first non-blank; else, just go back to the previous blank, then forward to the next non-blank.
- If any one of these searches failed, then position the cursor at the beginning of the current item, and update the current item record with ZEdUtil.Start - Item.

7.1.2.2.1.7 ZEdItem.BackW - exported

This procedure implements the ":" (move backward one "word") command, where a word is defined to begin with a space (or <CR>) character.

- If there's no text in the current item, then just Beep and exit
- Enter a loop for global variable, Iteration, counts, to call ZEdItem.BkWord for each word to move backward, incrementing the local iteration counter for each successful movement
- If the number of iterations accomplished is less than the number the user specified, then Beep

7.1.2.2.2 Procedures for Inserting characters

- ZEdItem.InsChr - exported
- ZEdItem.Extend - exported
- ZEdItem.InsBuf - exported

7.1.2.2.2.1 ZEdItem.InsChr - exported

This is the top-level routine for inserting characters into item text.

- Set Global boolean, Changed, to true
- Display the insertion message for the user with ZEdUtil.WrtMsg
- Invoke ZEdUtil.Insertion to do the actual character-by-character inserting
- Update the selection box boundaries with ZEdUtil.BoxBoundary
- Update the selection character (just in case) via ZEdUtil.UpdSelCh
- Display the default ZED message in the user display area with ZEdUtil.WrtMsg

7.1.2.2.2.2 ZEdItem.Extend - exported

This procedure implements the "x" (extend item) command, allowing the user to append characters to the end of the current item.

- If there's no string for the current item, then create a string record with a null string with NetMakeDel.CrFsP, and make global pointer variable, FirstStr, point to the new null string record
- Else, space over from the current position in the item to the end of the item using ZDspInc.DspCurR until it returns a false condition
- Invoke ZEdItem.InsChr to do the character insertion

7.1.2.2.2.3 ZEdItem.InsBuf - exported

This procedure implements the "↑" (insert characters from delete buffer) command.

- If there are no characters in the delete buffer, just Beep and exit
- Initialize
- For each character in the buffer, invoke ZEdUtil.Insert-Ch to insert the character after the current position and update the current position
- If the insertion terminated before all characters were inserted (via global boolean, AOK), then beep

- Update the selection box parameters with ZEdUtil.BoxBoundary

7.1.2.2.3 Procedures for modifying characters

- ZEdItem.Chg-Ch - exported
- ZEdItem.Replace - exported
- ZEdItem.NCvCase
- ZEdItem.CvCase - exported
- ZEdItem.Transpose - exported

7.1.2.2.3.1 ZEdItem.Chg - Ch - exported

This procedure implements the "c" (change character(s)) command.

7.1.2.2.3.1.1 If there's no text in the current item, just Beep and exit

7.1.2.2.3.1.2 If there's no character to the right of the current position (as determined by an unsuccessful ZDspInc.DspCurR), just Beep and exit

7.1.2.2.3.1.3 Replace cursor at original position (ZDspInc.DspCurL), set the global boolean, Changed, to true, initialize the iteration counter, i, to 1, turn the block cursor on (ZCanvas.CursorOn) and display the Change message

7.1.2.2.3.1.4 Enter the replacement loop for the iteration count:

- If the character was an escape (<INS>), then update the selection character, just in case, display the ZED message and exit
- If the character was NOT in [<BackSpace>, <Control-L>, <LF>, <CR>], then redisplay the old character, store the old character in the delete buffer via ZEdUtil.InsDelBuf, place the cursor at the character's position via ZDspInc.DspTxtPos, replace the character in the current item record, display the new character via ZCanvas.PutCh, move the cursor one position right via ZDspInc.DspCurR and increment the iteration counter by one
- Else, if the character is a <CR> or <LF> then use ZEdItem.DelChr to delete it and use ZEdUtil.Insert - ch to insert the new one, else simply flag an error

7.1.2.2.3.1.5 Update the selection character (ZEdUtil.UpdSelCh) in case it was changed

7.1.2.2.3.1.6 Display the default ZED message

7.1.2.2.3.2 ZEdItem.Replace - exported

This routine implements the "r" (replace character(s)) ZED command. The replaced characters are deleted, then an arbitrary number of characters are allowed to be entered to replace them.

- If there is no string pointed to in the current item, then just beep and exit
- If there is no text in the current item, then just beep and exit
- Delete the specified number of characters via ZDspInc.DspDelChr
- Allow the user to insert characters via ZEdItem.InsChr

7.1.2.2.3.3 ZEdItem.NCvCase

This is a support procedure used to help implement character case conversion. It will convert the case of iteration count characters starting from the current position in the current item.

Set up a loop for the iteration count to do the following:

7.1.2.2.3.3.1 If the current position is not at the end of the item, then:

- Obtain the character at that position
- Redisplay it with ZCanvas.PutCh
- Move and display the cursor at the current position
- If the character is alphabetic, convert its case, redisplay the new converted character with ZCanvas.PutCh, and insert it into the current item's string record at the current position

7.1.2.2.3.3.2 Move one position to the right (the next character) with ZDspInc.DspCurR

7.1.2.2.3.3.3 Increment the local iteration counter by one

7.1.2.2.3.4 ZEdItem.CvCase - exported

This routine implements the "v" (convert case) command. The default for this command is to convert a whole "word" if no iteration count is specified.

7.1.2.2.3.4.1 If the current row is row 1 in the item and there is no text pointed to, then just Beep and exit

7.1.2.2.3.4.2 If there is no "next" character (as determined by ZDspInc.DspCurR), then just Beep and exit

7.1.2.2.3.4.3 Move the cursor back; if there is an iteration count, then invoke ZEdItem.NCvCase, and exit

7.1.2.2.3.4.4 Conversion Loop

- Redisplay the current character with ZCanvas.PutCh;
- Move and display the cursor at the current position via ZDspInc.DspTxtPos;
- Convert the character's case, redisplay the new converted character with ZCanvas.PutCh, and insert it into the current item's string record at the current position;
- Move one position to the right (the next character) with ZDspInc.DspCurR;
- If the current position is past the end of the item, then just exit;
- Obtain the character at the new position.

7.1.2.2.3.5 ZEdItem.Transpose - exported

This routine implements the "t" (transpose two characters) command.

- If there are no characters to transpose, then just beep and exit
- Point to the current position in the current line
- If one beyond the current position is not beyond the end of the line, then interchange the character at the current position with the one immediately after it
- Else, just beep

7.1.2.2.4 Procedures for deleting characters

- ZEdItem.StarDelete
- ZEdItem.EraseToEnd
- ZEdItem.ReDspToEnd
- ZEdItem.Del←ALot←Ch
- ZEdItem.FastDeletion - exported
- ZEdItem.DelChr - exported
- ZEdItem.KillToChar - exported

- ZEdItem.KInsert - exported

7.1.2.2.4.1 ZEdItem.StarDelete

This procedure provides for deletion from the current cursor position to the end of the item, implementing the "*d" command. This assumes that the remainder of the current line has already been deleted by the calling routine.

- If there is no text in the current item, then just beep and exit
- Insert a <CR> into the delete buffer via ZEdUtil.InsDelBuf
- Point to the next string in the current item, with local variable, FsP
- For each next string in the item, invoke ZEdUtil.InsDelBuf to store each character in the string into the delete buffer, then, if there's a next string, insert a <CR> into the delete buffer. Point to the next string.
- If the original next string existed, then release the frame string record starting from that string via NetMakeDel.RelFsP
- Make the next string pointer point to NIL
- As a precaution, update the item's selection character

7.1.2.2.4.2 ZEdItem.EraseToEnd

This procedure is used to erase an item in the display window, from the current position on. It uses the fact that characters overwritten in XOR mode on the screen get erased.

- Invoke ZDspInc.DspErase with the current item pointer as the variable

7.1.2.2.4.3 ZEdItem.ReDspToEnd

This procedure is used to redisplay an item in the display window, from the current position on.

- Invoke ZDspInc.DspErase with the current item pointer as the variable

7.1.2.2.4.4 ZEdItem.Del - ALot - Ch

This routine is invoked when more than one line of characters must be deleted.

7.1.2.2.4.4.1 Decrement the iteration count by the number of characters left in the current line

7.1.2.2.4.4.2 Store the remaining characters on the current line in the delete buffer

7.1.2.2.4.4.3 Make the current string be the substring of the original string with PERQ - String.SubStr or a null string, if all the characters were deleted

7.1.2.2.4.4.4 If no next string, then exit

7.1.2.2.4.4.5 If Global Star is true, invoke ZEdItem.StarDelete, and exit

7.1.2.2.4.4.6 Enter loop to delete remaining characters on successive lines;

- Store a <CR> in the delete buffer with ZEdUtil.InsDelBuf, for the end of the previous line
- Set the limit for the next line's deletion of characters, based on the iteration count
- Store the appropriate number of characters of the current line in the delete buffer via ZEdUtil.InsDelBuf
- Decrement the iteration count by the number of characters just deleted
- Perform the actual delete on the current string via PERQ - String.Delete
- If there's a next string, point to it with the local string pointer, OFsP; else, signal the completion of the loop with local boolean done being set to true

7.1.2.2.4.4.7 Get rid of any null strings in the list via ZEdUtil.GetRidStrP

7.1.2.2.4.4.8 Redisplay the selection with ZEdItem.ReDsptoEnd and just in case, update the selection char with ZEdUtil.UpdSelCh.

7.1.2.2.4.5 ZEdItem.FastDeletion - exported

Upper level editing procedure for deleting more than one character.

- Erase the current line from the current position to the end via ZEdItem.EraseToEnd
- Decrement the iteration count by one
- invoke ZEdItem.Del - ALot - Ch to do the actual character deletion
- Place and display the cursor at its original position
- Update the selection box of the selection via ZEdUtil.BoxBoundary

7.1.2.2.4.6 ZEdItem.DelChr - exported

This is the top-level deletion routine for ZED.

- If there is no string in the current item pointer, just Beep and exit
- Set the global boolean, Changed, to true
- If the user has requested a "star" delete (to end of item) or the iteration count is greater than the length of the remaining text in the current string, invoke ZEdItem.FastDeletion to do the actual deletion; then, if the latter case (i.e., not "star"), delete one more character via ZEdUtil.Delete - Ch and decrement the delete buffer length by one; finally, just exit.
- Else, (all the characters to be deleted are on the current line) enter a loop for the iteration count, using ZEdUtil.Delete - Ch to delete individual characters, and incrementing the local iteration counter by one each pass; then, update the selection box via ZEdUtil.BoxBoundary, the selection character with ZEdUtil.UpdSelCh, and turn the cursor back on.

7.1.2.2.4.7 ZEdItem.KillToChar - exported

This procedure implements the "k" (delete up to specified character) command.

- If there's no text, just Beep and exit
- Save current item information, get the character and initialize
- Invoke ZEdItem.Find - Ch iteration times
- Save the "found" row and column information for later reference and restore the original current item information
- If the found character is in the same row as the current row, then just store and delete the appropriate number of characters, move the cursor, display the remainder of the line with ZDspInc.DspLin, update the selection character if necessary and exit.
- Otherwise, go through each line, storing and deleting characters with ZEdUtil.InsDelBuf and PERQ - String.Delete until the specified row and column are reached
- Restore the original current item information, and redisplay the entire selection, clean up by deletin any extra characters and update the selection character

7.1.2.2.4.8 ZEdItem.KInsert - exported

This procedure implements the "m" (kill to character, then insert - "munch") command.

- If there's no text in the current item, just beep and exit
- Invoke ZEdItem.KillToChar to delete the intended characters
- If the kill to character was successful, then invoke ZEditem.InsChr to allow the user to insert characters at the point of the deletion

7.1.2.2.5 Procedures to find characters within an item

- ZEdItem.LastOcc
- ZEdItem.StrSearch
- ZEdItem.Find←Str - exported
- ZEdItem.FindCR
- ZEdItem.Find←Ch
- ZEdItem.GetSChr
- ZEdItem.CharSearch - exported

7.1.2.2.5.1 ZEdItem.LastOcc

Low-level support for string searching. This will return an index to the rightmost (last) match in a string, starting from the current position, as opposed to the first (leftmost).

7.1.2.2.5.1.1 Enter a loop to do the following:

- Find an occurrence of the match string in the source string using the NetString.AnyPos (case-insensitive search) utility
- If one is found, increment the starting position accordingly, else, set the ending condition of the loop true
- If there's enough of the source string left, remove the part already scanned with PERQ – String.SubStr, else, set the ending condition true

7.1.2.2.5.1.2 Return the last index of the found string

7.1.2.2.5.2 ZEdItem.StrSearch

This is the basic string matching routine for the current selection, starting from the current position.

7.1.2.2.5.2.1 Save the current item information and initialize, then enter the main loop to do the following:

- If the first time through the loop, obtain the appropriate substring of the current line to search, and search it with ZEdItem.LastOcc or NetString.AnyPos; else, just do the appropriate search on the entire line
- If a match was found, then update the Current Item column to reflect the found position, move the cursor to the matched string via ZDspInc.DspTxtPos and exit
- Otherwise, go to the next (previous, if searching backwards) string in the current item if possible, and repeat

7.1.2.2.5.2.2 If no match was found, restore saved current item information

7.1.2.2.5.3 ZEdItem.Find - Str - exported

This procedure implements the "f" (find string) command.

- If there's no string to search in the current item, just Beep and exit
- Initialize, then display "enter search string" message, and obtain the user's input string via ZEdUtil.ReadStr
- Store the user's new search string in Global variable, PatStr
- Initialize and enter the search loop to do the iteration count, calling ZEdItem.StrSearch each pass, and incrementing the local iteration counter
- Reset the global boolean, backsearch, if necesary
- Move the cursor the the appropriate place, stored in CItem, via ZDspInc.DspTxtPos
- Redisplay the default ZED message with ZEdUtil.WrtMsg

7.1.2.2.5.4 ZEdItem.FindCR

This routine searches for an (implicit) <CR> in the current item.

- If doing a backwards search (Global BackSearch is true) then if there is a previous string in the current item, point to it, move up a row, move the current column to the end of this new string, display the cursor there, and exit
- Else, try to move the cursor one position to the right with ZDspInc.DspCurR
- If this was successful, then move the current column to one beyond the end of the current line, and display the cursor there

7.1.2.2.5.5 ZEdItem.Find - Ch

The general character searching routine for the current item.

- If the search character is a <CR>, invoke ZEdItem.FindCR and exit
- Save current item information and initialize for loop
- Enter search loop:
- Move the cursor on character in the appropriate direction, depending on the direction of the search
- If not at the end of the text, then if there's a match, indicate so in the var boolean, Found, move the cursor there, and exit; else repeat the loop
- If no match was found, restore the old current item information, and redisplay the cursor

at the original position via ZDspInc.DspTxtPos

7.1.2.2.5.6 ZEdItem.GetSChr

This is a support routine for the character searching command, to obtain the character to search for from the user.

- Create the "enter target character" message
- Display it in the user display line via ZEdUtil.WrtMsg
- Move the cursor to the input position, via ZDisplay.DspPos
- Obtain the input character via ZCanvas.RdKbd
- Redisplay the default ZED message

7.1.2.2.5.7 ZEdItem.CharSearch - exported

This routine implements the "s" (search for character) command.

- If there is no string to search in the current item, then just Beep and exit
- Obtain the character to search for via ZEdItem.GetSChr
- Initialize, then enter the search loop for Iteration times, calling ZEdItem.Find-Ch, incrementing the local counter by one each time a match is made
- Display the default ZED message

7.1.2.2.6 Procedures for justifying text within an item

- ZEdItem.TestBlank
- ZEdItem.Rebuilt
- ZEdItem.RmExtraBlanks
- ZEdItem.Decompose
- ZEdItem.CkTotalLength
- ZEdItem.GetMaxLen
- ZEdItem.Bomb
- ZEdItem.Shuffling
- ZEdItem.Justify - exported
- ZEdItem.SmallJ - exported

7.1.2.2.6.1 ZEdItem.TestBlank

This is a support function. It returns true if the string argument consists of all blanks.

- For each character in the string, check if it is a non-blank (space). If so, return false
- If the loop for the string checking has been completed, then return true, since all the characters in the string were blank.

7.1.2.2.6.2 ZEdItem.Rebuilt

This procedure reformats a long string (in BufStr type record) into a returned frame string record (linked list of strings of length input parameter MaxLen).

Set up a the loop to remove substrings from the input string:

- Advance the pointer from the previous position to up to MaxLen characters further in the input string
- Strip off any trailing blanks by retreating the pointer back past them
- Move these up to MaxLen characters into a local string variable
- Create a string record via NetMakeDel.CrFsP for this string
- If it's the first pass, make the var pointer variable, NewStrP, point to this newly created record; else, add this newly created record to the previously created list of strings
- Store copy of the new string record and move the offset into the input string appropriately

7.1.2.2.6.3 ZEdItem.RmExtraBlanks

This procedure replaces all occurrences of multiple adjacent spaces in the var string with a single space.

- Invoke NetString.Strip to remove any blanks from the ends of the input string
- Examine the input string downwards from the end for occurrences of two adjacent blanks
- If the second character is a blank, use PERQ - String.Delete to remove the first blank from the string

7.1.2.2.6.4 ZEdItem.Decompose

This routine does the opposite of ZEdItem.Rebuilt: it forms a continuous long string (of type BufStr) from a Frame String list.

7.1.2.2.6.4.1 Initialize the string buffer to zero length

7.1.2.2.6.4.2 For as long as there are more "lines" in the string record, do:

- If the current "line" in the string record is all blanks, then insert a single null character (as a flag) in the string buffer

- Else, remove multiple adjacent blanks from the current "line" with ZEdItem.RmExtraBlanks, insert all the characters from the current "line" into the string buffer, and tack on a blank in the string buffer

- Point to the next line in the string list

7.1.2.2.6.4.3 Remove the last blank character added in the string buffer

7.1.2.2.6.5 ZEdItem.CkTotalLength

This procedure returns true if there is enough room in the frame for the passed Frame String record.

- Initialize, and point to the "next" string in the passed frame string record

- Add up the total number of rows from the current "line" there are in the Frame String record

- Add to this, the row position of the current selection (passed as parameter Row) and the number of rows "down" in the current selection we already are

- If this sum is less than the length of the frame, minus 1, then the selection can fit, so return true

7.1.2.2.6.6 ZEdItem.GetMaxLen

This procedure prompts the user for right margin to use, and returns this information as a maximum length of string to use for the justification of the current item. The process is slightly different for Text and Title than for Options or Local Pads, since there is a reasonable default for the former.

- If the current item is Title or Text, display the prompt in the user display line with ZEdUtil.WrtMsg, and obtain the response (default "y") with ZCanvas.RdKeyEv. If a positive response is received, set the var MaxLen to 80 minus the current position and exit.

- Display the prompt for the user to indicate where he/she wants the right margin with the mouse, and wait for a mouse click in response, via ZCanvas.RdKeyEv. Adjust the user-indicated mouse column position to be inside the frame window, and return a value of MaxLen equal to the mouse column position minus the current position within the item plus one.

- Redisplay the ZED default message

7.1.2.2.6.7 ZEdItem.Bomb

This procedure merely displays one of three error messages, depending on its argument. It was created to localize these three messages for access by the upper level item justifying routines.

- Beep the user, indicating a problem
- If argument is 1, display "no need to justify" message
- If argument is 2, display "Abort" message
- If argument is 3, display "line length too short for justify" message
- Set Global boolean, ChgMsg, to true, to force redisplay of ZED default message at the command level

7.1.2.2.6.8 ZEdItem.Shuffling

This is the main justify routine for selections. Text and Title are handled separately in the calling routine. It re-forms the first line as a line of length MaxLen, then indents the remaining lines three spaces in.

- Do a ZEditem.Decompose/Rebuilt pair on the selection text, starting at the first line of the text. This will correctly format the first line of the text to a length of MaxLen.
- If there are next line(s), then re-execute the ZEdItem.Decompose/Rebuilt pair on them, this time with a length of three less than MaxLen
- If the justified selection won't now fit, display an error message via ZEdItem.Bomb, redisplay the selection with ZEdUtil.ReDsp, and exit with no permanent changes
- Otherwise, prepend three blanks onto each of the remaining lines with PERQ – String.Concat to indent them properly
- Reset the current item record, update the selection box information with ZEdUtil.BoxBoundary, and redisplay the selection with ZEdUtil.ReDsp.

7.1.2.2.6.9 ZEdItem.Justify - exported

This procedure implements the "J" (justify whole item) command.

- If there's no text in the current item, display error message via ZEdItem.Bomb and exit
- If there's only one line of text in the current item, display "no need" message via ZEdItem.Bomb and exit
- If the user specifies a string length less than 15 display error message via ZEditem.Bomb and exit
- If the current item is an option or a local pad, then invoke ZEdItem.Shuffling to do the justifying, and just exit

- Else, invoke the ZEdItem.Decompose/Rebuilt pair, and check to make sure the newly justified item still fits in the window. If so, release the old global first string pointer, FirstStr, and make it point to the newly formatted one, and update current item information; else, display error message via ZEdItem.Bomb, and release the new string pointer.

- Redisplay the item via ZEdUtil.ReDsp

7.1.2.2.6.10 ZEdItem.SmallJ - exported

This procedure implements the "j" (justify item from current position) command.

7.1.2.2.6.10.1 If there's no text, or there's only one line and we're at the end of it, just ZEdItem.Bomb and exit

7.1.2.2.6.10.2 Get the MaxLen for the justification, via ZEdItem.GetMaxLen, or default to selection box width, if we're at any other row than the first; if the MaxLen is too small, ZEdItem.Bomb and exit.

7.1.2.2.6.10.3 If MaxLen is less than the current position, process, and exit

- Update globals with ZEdUtil.Start – Item

- Do one ZEdItem.Decompose/Rebuilt pair for the first row, then again, starting from the new second row, with a new MaxLen, decremented by three if the item is an option or local pad

- If the newly justified item no longer fits, as determined by ZEdItem. CkTotalLength, ZEdItem.Bomb, release the local string record, redisplay the item and exit.

- Link in the remaining text of the item

- If the item is an option or local pad, pad the remaining lines of the item with three blanks to line up the text on the left

- Make the current string the first string of the item

- Update the selection box via ZEdUtil.BoxBoundary

- Redisplay the whole item with ZEdUtil.ReDsp

7.1.2.2.6.10.4 If MaxLen is greater than the current text length, and there's no next string, ZEdItem.Bomb and exit

7.1.2.2.6.10.5 Preprocess

- Set local boolean, Done, to false
- Save the current column position, and reposition to the beginning of the current string for processing
- Save the pointer to the previous string in the item
- Erase the remainder of the item from the (new) current position on with ZEdItem.EraseToEnd

7.1.2.2.6.10.6 If we're currently on row 1 of the item, then process the whole item

- Execute one ZEdItem.Decompose/Rebuilt pair
- If there's more lines in the justified item, then execute another ZEdItem. Decompose/Rebuilt pair, with a MaxLen decremented by three if the current item is an option or local pad
- If the newly formatted item cannot fit in the frame display window, as determined by ZEdItem.CkTotalLength, ZEdItem.Bomb and exit
- Release the old first string record, pointed to by FirstStr
- Make the new current string the first string in the justified item
- Reset FirstStr to the current string
- Make local next string pointer, NStrP, point to the next string in the newly justified item for later processing

7.1.2.2.6.10.7 Else process from the current position

- Make local string pointer, NStrP, point to the current string in the item
- Execute a ZEdItem.Decompose/Rebuilt pair, with MaxLen decremented by three if the current item is an option or local pad
- If the newly justified item will not fit in the frame display window, as determined by ZEdItem.CkTotalLength, then ZEdItem.Bomb and exit
- Release the saved string record from the original item, and update it with the new one, at the current position
- Update the current string pointer with the new one
- Make sure that the local string pointer, NStrP, also points to the new current string, for the following processing

7.1.2.2.6.10.8 If the item is an option or local pad, pad the text with three blanks; Redisplay the item and the cursor and update the selection box with ZEdUtil.BoxBoundary

7.1.2.2.7 Miscellaneous procedures

- ZEdItem.CTL+U - exported
- ZEdItem.ClrBuf - exported
- ZEdItem.AltSelChr - exported

7.1.2.2.7.1 ZEdItem.CTL-U - exported

This routine implements the "<Control-U> | <OOPS-key>" command, to "undo" the last changes to the current item. No text deleted by this procedure is put into the delete buffer.

- If there is no text in the current item (i.e., current row is row 1, and there is no text on this line), then just Beep and exit
- Erase the item via ZEdUtil.Erase (which simply overwrites the item with itself in XOR mode)
- If there were any changes made to the item (Global boolean changed is true), then release the top of the string list via NetMakeDel.RelFsp, and reset the current selection record position and selection box items to their original values (which were saved via a previous ZEdUtil.ItemInit call!)
- Redisplay the original current item via ZEdUtil.ReDsp

7.1.2.2.7.2 ZEdItem.ClrBuf - exported

This routine implements the " # " (clear the delete buffer) command.

- Set the length item of the Delete Buffer record to zero

7.1.2.2.7.3 ZEdItem.AltSelChr - exported

This procedure implements the "z" (Modify selection character) command.

- If the current item is frame text or title, just Beep and exit
- Form the "enter new selection character" message, with the current selection character as the default
- Display it with ZEdUtil.WrtMsg, and read in the user's response with ZCanvas.RdKbd (Should be RdKeyEv, to allow mouse click for default)
- If the input character is not in [<CR>, <LF> or Escape (INS)] then if it was a Control-Q (for quote next character) read the next character via ZCanvas.RdKbd. Assign the character to the current selection's selection character, and indicate that a change was made, via

Global boolean, changed.

- Display the default ZED message

7.1.3. ZED Support Modules

These modules provide either ZED-wide definitions and global variables, or low-level utilities for use by the command implementing routines:

7.1.3.1 ZEdDefs - ZED TypeDefs and Global Variables

This module serves to define some of the record structures in ZED, and also to declare most of the Global variables within ZED. Since these globals are referenced throughout the ZED code, it is important to become familiar with them early on.

7.1.3.1.1 Selection pointer variables

CurSelP	SelPTyp; Points to the current selection in the edited frame, as moved to or pointed to with mouse - All within item routines refer to this variable
NewSelP	SelPTyp; General purpose variable, typically used during new item creation
SaveSelP	SelPTyp; Another general purpose variable, typically used to store old information in case some changes are not made permanent

7.1.3.1.2 Frame pointer variable

EdFP	FPTyp; Points to frame currently being edited; Note: this is not cleared when ZED is exited via the "@" or "h" commands, so that ZED can resume editing the last frame even if it is not currently displayed

7.1.3.1.3 Frame string pointer variable

FirstStr	FsPTyp; Points to the "first" string in the current item. Useful for quick reference back to the "beginning" of the current item

7.1.3.1.4 ZED-defined record variables

CItem	TxtPosTyp; Refers to the specific current position within the current item. Contains relative row and column info, and current string
TypeItem	ItemType; Keeps track of what the current item is
BufP	BufStrP; Points to the delete buffer
SelBuf	Selections; Record holding the selected text and related information - not Current Used

7.1.3.1.5 Other typed variables

Akey	CanvEvTyp; Used for holding user-supplied responses to ZED prompts, via keyboard or mouse

CmdType, SaveCmdType	(MouseCmd, KbdCmd, KbdErr); An enumerated type used for recording statistics regarding type of input user gave
OldCurs	CursorType; Used for saving a cursor type the user had prior to entering the editor

7.1.3.1.6 String and character variables

ChrR	char; Used to store character information input by user for subsequent processing, usually command parsing
Msg0	string; Global message variable, used for constructing prompt messages to be displayed in user display line
EdMsg	string; This stores the default ZED message, "Type h for *Edit* help, e to exit"
PatStr	string; Stores user-input string for pattern matching in some of the string-search routines

7.1.3.1.7 Integers

OrigC0, OrigC1	integer; Saves old current item width parameters
OrigL0, OrigL1	integer; Saves old current item height parameters
OrigRow, OrigColumn	integer; Saves old current position information
Iteration	integer; Stores user-input iteration count
OldWindow	integer; Stores the previous window number, for exiting/reentering ZED after "@" or "h" Commands
LenChFrame	integer; Height in characters of current frame Either 24, for "normal" sized frame, or 49 for "big" frame

7.1.3.1.8 Booleans

DontBufferIt	True if deleted text not to be stored in delete buffer
Star	True if user input an asterisk ("*") prior to command
Changed	True if item was changed
Back	True if word cursor motion to be backwards
NewSel	0
MouseUsed	True if mouse button was clicked in response to prompt
Exiting	True if command leads to leaving ZED: "e","q","@", "h"
BackSearch	True if char/string searches to be performed backwards
XCHG	True if any part of frame has been modified
Xtend	True if insertion is in Extend-Options ("X") mode

BackSpace	True if cursor movement is to be backwards
ChgIteration	True if user entered an iteration count
AltCursorOn	True if one of the alternate cursors was on prior to ZED
Guest	True if current user is "Guest"

7.1.3.2 ZDspInc - Low-Level Display Utilities for ZED

This module contains those routines which manipulate the cursor and item display in the current window while in the editor, ZED.

7.1.3.2.1 Cursor and Item display routines

- ZDspInc.IniTxtPos - exported
- ZDspInc.DspTxtPos - exported
- ZDspInc.DspLin - exported
- ZDspInc.ClrLin - exported
- ZDspInc.DspErase - exported

7.1.3.2.1.1 ZDspInc.IniTxtPos - exported

This procedure initializes the current text item record to point to the beginning of the specified item. It is normally called with global CItem as the var parameter.

- Set the current item selection pointer to point to the current selection
- Initialize the relative row and column information to the beginning of the item (relative row 1 and column 1)
- Set the current string pointer to point to the passed string within the item (normally, global frame string pointer, FirstStr)

7.1.3.2.1.2 ZDspInc.DspTxtPos - exported

Given the current item information, this routine will display the cursor at the specified position within the item.

- Invoke ZDisplay.DspPos to display the cursor. The arguments passed are calculated as the row of the item plus the relative row within the item minus 1 (since relative row starts at 1, not 0), and similarly for the column.

7.1.3.2.1.3 ZDspInc.DspLin - exported

This routine will display the (remainder of) the current line within the current text item.

- Invoke ZCanvas.PutCh to display the characters of the current line from the starting position within the line to the end of the line of text.
- If there is an external terminal, form a string with the remaining text in the current line and output it to the external terminal with ZIO.SendStr
- Invoke ZDspInc.DspTxtPos to redisplay the cursor at its proper position

7.1.3.2.1.4 ZDspInc.ClrLin - exported

This routine will clear the (remainder of) the current line within the current text item. It uses the fact the the character display mode is XOR, so that redisplaying a character will erase it.

- Invoke ZCanvas.PutCh to redisplay the characters of the current line from the starting position within the line to the end of the line of text.
- Do the same with ZIO.SendChar to output to an external terminal
- Invoke ZDspInc.DspTxtPos to redisplay the cursor at its proper position

7.1.3.2.1.5 ZDspInc.DspErase - exported

This routine will display (erase, since character display is in XOR mode, the (remainder of the) current text item. It is up to the caller to determine which he/she wishes to do, display or erase the item.

- Display the cursor at the current position via ZDspInc.DspTxtPos
- Display (erase) the current line with ZDspInc.DspLin
- Save the old current item information
- Do the ZDspInc.DspTxtPos/DspLin pair for the remaining lines in the current item, always from the beginning of each line
- Restore the saved current item row, column and current string information
- Redisplay the cursor at the (original) current position with ZDspInc.DspTxtPos.

7.1.3.2.2 Cursor movement routines

- ZDspInc.DspCurR - exported
- ZDspInc.DspCurL - exported
- ZDspInc.DspCurU - exported

- ZDspInc.DspCurD - exported

7.1.3.2.2.1 ZDspInc.DspCurR - exported

This procedure will attempt to move the cursor (and current position) one character to the right within the current item. It will return false if the move cannot be done.

- Initialize var boolean, OK, to True
- If the current position is not at the end of the current line, then just increment the current column position by one, and redisplay the cursor there with ZDspInd.DspTxtPos
- Else, if there is NOT a next string, set OK to false, and exit; else advance to the next line within the current item, increment the row position by one, and reset the column to column 1. Finally, redisplay the cursor at the new position with ZDspInc.DspTxtPos.

7.1.3.2.2.2 ZDspInc.DspCurL - exported

This procedure will attempt to move the cursor (and current position) one character to the left within the current item. It will return false if the move cannot be done.

- Initialize var boolean, OK, to True
- If the current position is not at the beginning of the current line, then just decrement the current column position by one, and redisplay the cursor there with ZDspInd.DspTxtPos
- Else, if there is NOT a previous string, set OK to false, and exit; else go back to the previous line within the current item, decrement the row position by one, and reset the column to the end of the (previous) line of text. Finally, redisplay the cursor at the new position with ZDspInc.DspTxtPos.

7.1.3.2.2.3 ZDspInc.DspCurU - exported

This procedure will attempt to move the cursor one line upward within the current item, staying in the same column. It will return false if the move cannot be done.

- Initialize var boolean, OK, to true
- If there is no previous string in the current item, then Beep and set OK to false
- Else, decrement the current row by one, and make the current item's current string pointer point to the previous string. If the length of this new string is less then the old column position, update the column position to be at the length of the new (current) string. Finally, redisplay the cursor within the text with ZDspInc.DspTxtPos.

7.1.3.2.2.4 ZDspInc.DspCurD - exported

This procedure will attempt to move the cursor one line downward within the current item, staying in the same column, if possible. It will return false if the move cannot be done.

- Initialize var boolean, OK, to true
- If there is no next string in the current item, then Beep and set OK to false
- Else, increment the current row by one, and make the current item's current string pointer point to the next string. If the length of this new string is less then the old column position, update the column position to be at the length of the new (current) string. Finally, redisplay the cursor within the text with ZDspInc.DspTxtPos.

7.1.3.2.3 Character and line insertion/deletion routines

- ZDspInc.LeadingBlks - exported
- ZDspInc.GPushWord
- ZDspInc.DspInsChr - exported
- ZDspInc.DspInsLin - exported
- ZDspInc.DspDelChr - exported
- ZDspInc.DspDelLin - exported

7.1.3.2.3.1 ZDspInc.LeadingBlks - exported

The purpose of this procedure is to count the number of leading blanks in the string passed to it.

- Initialize the var integer, NumBlks to 0, and the iteration counter to 1
- Obtain the length of the input string via PERQ - String.Length
- Check each character in the input string, starting from the first character; as long as the characters are blanks, increment the blank counter, otherwise, set the loop terminating condition.

7.1.3.2.3.2 ZDspInc.GPushWord

This is a utility procedure invoked while inserting characters in a line. It's purpose is to search the input string backwards from its end for the first blank, whose position in the string is LESS than parameter, MaxLen. It returns an index to this position. The effect is to point to the blank just before the last word in the line which will get "pushed down" during insertion.

- Get the length of the input string
- Starting at the end of the input string, search for the first blank

- If a blank is found, then set var integer, ptr, to the index of the blank, and set local boolean (and loop terminating condition), found, to true
- Check to see if the pointer is within MaxLen of the beginning of the string; if NOT, reset local boolean, found, to false in order to repeat the process.

7.1.3.2.3.3 ZDspInc.DspInsChr - exported

This procedure will insert a character at the current position in the text of the current item. It is complicated by the fact that the addition of one character in a line of text may make that line too long to fit in the available space. If that is the case, this procedure will find the last "word" of the current line, append it to the beginning of the next line, and so on, until the end of the item. In some cases, a new line may have to be created in which to place the last word of the item. Proper line indentation is maintained through out the pushing down of words.

The code for this procedure is lengthy and convoluted.

7.1.3.2.3.4 ZDspInc.DspInsLin - exported

This procedure is used to insert a new line of text into an existing item.

- Make local pointer variable, OStrP, point to the first line of item and count the number of lines currently in the item
- If there are too many lines already, beep and exit
- Save original current item information
- If there are next lines in the item, clear each one out via ZDspInc.ClrLin and then redisplay them one line downward with ZDspInc.DspLin
- Make current string pointer point to top line in item and obtain the number of leading blanks in that first line
- Insert that many blanks at the beginning of the new string, to make it line up with preceding lines
- Create a new string pointer with the given text, then add it to the list for the current item
- Restore updated current item information, display the new line with ZDspInc.DspLin, and redisplay cursor at updated location

7.1.3.2.3.5 ZDspInc.DspDelChr - exported

This procedure will delete a character at the current position in the current item.

- If the current position is at the end of the current line (i.e., is at the <CR> character ending the line), invoke ZDspInc.DspDelLin to delete the current line, starting from that ending position

- Else, delete the current line with ZDspInc.ClrLin, delete the appropriate character via PERQ – String.Delete, and redisplay the modified line with ZDspInc.DspLin

7.1.3.2.3.6 ZDspInc.DspDelLin - exported

This procedure is used to "delete" the current line. In reality, it tries to delete the (implied) <CR> at the end of the current line, concatenate the next line to the current one and redisplay.

- If there's no next line, then simply exit
- Calculate the number of extra characters that will fit on the current line into local variable, LineLen, and save original current item information
- Erase lines in the current item, starting with the current line, with ZDspInc.ClrLin
- Concatenate the (remainder of the) current line with the next line with a connecting blank; store the length of the original line plus the blank
- If the length of the concatenated string is greater than LineLen, restore things with GPushWord
- Else, make the concatenated string the new string at the current position, and delete the extra string record from the list
- Redisplay the item's lines and restore/update the current item information

7.1.3.3 ZEdUtil - Low-Level utility routines for ZED

7.1.3.3.1 Display utilities

- ZEdUtil.DspFPEd - exported
- ZEdUtil.DspEditorHdAndTail - exported
- ZEdUtil.DspRowCol - exported
- ZEdUtil.DspItem - exported
- ZEdUtil.Erase - exported
- ZEdUtil.ReDsp - exported
- ZEdUtil.GErase
- ZEdUtil.GReDsp

7.1.3.3.1.1 ZEdUtil.DspFPEd

This procedure simply displays the frame stored in the editor frame record pointed to by EdFP.

- Set global SigNoClear to false
- Call ZDisplay.DspF to display the frame body
- Call ZDisplay.DGPads to display the associated global pads of the frame

7.1.3.3.1.2 ZEdUtil.DspEditorHdAndTail

This routine displays the "*Edit*" string in the upper right corner of the edit frame window, and the ZED "global pads" on the bottom line of the window.

- Put the character display in replace mode, via ZCanvas.SetChFunc
- Place the cursor at 1,59 via ZDisplay.DspPos, form the string, and display the string via ZCanvas.PutStr
- Position the cursor at the beginning of the bottom line, and clear that line with ZDisplay.ClrEoLn
- Form the "global pads" string and display it with ZCanvas.PutStr
- Use ZEdUtils.WrtMsg to display the default ZED message

7.1.3.3.1.3 ZEdUtil.DspRowCol

This is a support utility for the Positioning command (See ZEdUtil.Positioning) to display the X-Y coordinates of the position of the mouse cursor in the lower right corner of the window while repositioning items.

- Turn the cursor off with ZCanvas.CursorOff
- Get the current mouse position with ZCanvas.GetPosCh
- Put the character display into replace mode with ZCanvas.SetChFunc
- Place the cursor at (UsrDispLine, 71) with ZDisplay.DspPos
- Form the string for row position and display it with ZCanvas.PutStr
- Form the string for column position and display it with ZCanvas.PutStr
- Make sure the mouse cursor is positioned there with ZCanvas.SetPosCh
- Reset the display function and redisplay the cursor

7.1.3.3.1.4 ZEdUtil.DspItem

This is a general routine to either display or clear the current item.

- Invoke ZEdUtil.Start - Item to set the current item variables to point to the beginning of the item

- For each line in the item, display the cursor at the beginning of the line, invoke ZDspInc.ClrLin if we're erasing the item, or ZDspInc.DspLin otherwise. Note: For each line, ZCanvas.PollCanvas is called to queue up any user-input characters.

- Reposition things at the beginning of the item via ZEdUtil.Start - Item

7.1.3.3.1.5 ZEdUtil.Erase

This routine will erase the current item pointed to by CItem.

- Set boolean erasing to true for the following call

- Invoke ZEdUtil.DspItem with argument true, so as to erase the current item

7.1.3.3.1.6 ZEdUtil.ReDsp

This routine will display the current item pointed to by CItem.

- Set boolean erasing to false for the following call

- Invoke ZEdUtil.DspItem with argument false, so as to display the current item

7.1.3.3.1.7 ZEdUtil.GErase

This utility is used by GPOS to erase all the options on the frame being edited by re-writing them in XOR mode.

- Point to the first option in the option list

- For each option in the option list, redisplay the option text with ZDisplay.DspFsp, then point to the next option on the list until there are no more

7.1.3.3.1.8 ZEdUtil.GReDsp

This utility is used by GPOS to redisplay all the options on the frame being edited by writing them in XOR mode.

- Point to the first option in the option list

- For each option in the option list, display the option text with ZDisplay.DspFsp, then point to the next option on the list until there are no more

7.1.3.3.2 Screen/keyboard/mouse I/O Utilities

- ZEdUtil.WrtMsg - exported
- ZEdUtil.BEEP - exported - Simply invoke ZCanvas.BEEPCanvas to sound the "bell"
- ZEdUtil.ReadStr - exported
- ZEdUtil.CheckCommand - exported

7.1.3.3.2.1 ZEdUtil.WrtMsg

This routine displays the given string in the User Display line of the frame being edited.

- Turn off the cursor via ZCanvas.CursorOff
- Put the character display into replace mode via ZCanvas.SetChFunc
- Get and save the current cursor position with ZCanvas.GetPosCh
- Put the cursor at the User Display line with ZDisplay.DspEnd and clear that line with ZDisplay.ClrLine
- Invoke ZCanvas.PutStr to print out the string, then if there's an external display device, call ZIO.SendStr to output the string to it (not necessary in newer versions, since ZCanvas.PutStr handles that)
- Restore the cursor to it original position with ZCanvas.SetPosCh, return the display function to XOR mode and redisplay the cursor with ZCanvas.CursorOn

7.1.3.3.2.2 ZEdUtil.BEEP - exported - Simply invoke ZCanvas.BEEPCanvas to sound the "bell"

7.1.3.3.2.3 ZEdUtil.ReadStr

This routine allows string input, terminated by <CR>, <LF> or Escape (<INS>) from the user. It also handles backspace properly.

7.1.3.3.2.3.1 Display the cursor at the proper column in the User Display Line and initialize

7.1.3.3.2.3.2 Read in a character with ZCanvas.RdKbd

- If the character is a backspace, do the backspace stuff (move back one space, redisplay character to erase it, and move back again)
- Else, if it is a Control-L, then Beep
- Else, if the input line is too long, then display a message for the user to shorten it with BackSpace

- Else put the character into the Var String, Str, and display it
- Read another character for the next pass until a <CR>, <LF> or <INS> is typed

7.1.3.3.2.3.3 Adjust the Var string to the proper length

7.1.3.3.2.4 ZEdUtil.CheckCommand

This routine is used to interpret a mouse button click on the global pads line for a ZED command. It is hard coded with the current positions of each of the abbreviated ZED commands, so it simply compares the X-position of the mouse when the Mouse Event occurred with each of the positions of the commands, until a match (or almost match) is found. The corresponding command character is returned to th e calling routine via Var character, Ch.

7.1.3.3.3 Command Implementing procedures

- ZEdUtil.Start-Item - exported
- ZEdUtil.FrameText - exported
- ZEdUtil.InsWhoWhen - exported
- ZEdUtil.ItemInfo - exported
- ZEdUtil.Positioning - exported
- ZEdUtil.GPOS - exported

7.1.3.3.3.1 ZEdUtil.Start-Item

This routine implements the "I" (go to the beginning of the current item) command. It is also used as a utility to initialize the cursor to the beginning of the current item.

- Set the current item's current relative row and column position to 1
- Make the current item's current string pointer point to the first string of the item
- Display the cursor at the (new) current position within the current item with ZDspInc.DspTxtPos

7.1.3.3.3.2 ZEdUtil.FrameText

This procedure implements the "L" (go to the beginning of the frame text) command. It is also used as a utility to point the current item to the frame text of the frame being edited.

- If the current item is the frame text, then just point to the beginning of it via ZEdUtil.Start-Item
- Else, invoke ZEdUtil.Save-Chg, make the current item the frame text and init with

ZEdUtil.ItemInit

7.1.3.3.3.3 ZEdUtil.InsWhoWhen

This routine implements the "w" (insert date/time/current user stamp) command.

- If there is no current string pointed to, create one via NetMakeDel.CrFsP
- Get the current user name with NetHandl.GldUsr and start the working string with it
- Get the current date with BaseLib.GDatStr and append it to the working string
- Get the current time with BaseLib.GTimStr and append it to the working string
- Surround the working string with square brackets
- For each character in the working string, display the character with ZDspInc.DspInsChr and move the cursor one position to the right with ZDspinc.DspCurR
- Update the selection character if necessary via ZEdUtil.UpoSelCh

7.1.3.3.3.4 ZEdUtil.ItemInfo

This procedure implements the "?" (display information about the current item) command.

- Initialize the working string with text
- Get the current row and column information from the current item record, and append them to the working string
- For Title or Text, append the appropriate string label for the item type, and display the working string via ZEdUtil.WrtMsg and exit
- For Options and Local Pads, continue by appending the appropriate string label for them
- Append the selection's selection character, next frame information, and selection action presence/absence information to the working string
- Display the working string with ZEdUtil.WrtMsg

7.1.3.3.3.5 ZEdUtil.Positioning

This procedure implements the "p" ((re) position current item) command.

- Display the move prompt message
- Initialize variables, and display current item row and column information
- Read a key event to start the move loop with ZCanvas.RdKeyEv
- For each character read in, other than a terminator (Escape or period) do a Case on the character, invoking the appropriate ZEdUtil.Move* routine, or invoking ZEdUtil.Drag to

handle moving the item with the mouse

- Update the row and column, display the new row and column information with ZEdUtil.DspRowCol and read in the next character to continue the loop

- If the item was an option or a local pad, then some adjustment of its position in the appropriate selection list may be necessary. Do so by first deleting the selection from the selection list via NetMakeDel.DelSelL, and re-insert into the list with NetOption.InsOptF or InsPadF as appropriate.

7.1.3.3.3.6 ZEdUtil.GPOS

This routine implements the "P" ((re) position the entire group of options) command.

- If there are no options, or only one option, then just Beep and exit

- Display the positioning message, invoke ZEdUtil.GetCorners to get the "super" selection box for all the options, and read in a character with ZCanvas.RdKbd (NOTE: No mouse use allowed here)

- Do a case for the input character to determine the amount and direction of movement - store this information in local variables, DRow and DCol

- Erase, then redisplay at the new positions, all the options, with ZEdUtil.GErase, and ZEdUtil.GReDsp, respectively, and update the "super" selection box parameters accordingly

- Read the next character until a terminator (Escape or period) is read in

- Initialize with ZEdUtil.ItemInit and Start←Item and ZDspInc.DspTxtPos for the current item

7.1.3.3.4 Insert/Delete utilities

- ZEdUtil.InsBlank

- ZEdUtil.InsKcrKlf (internal procedure to Insert←Ch)

- ZEdUtil.Insert←Ch - exported

- ZEdUtil.insertion - exported

- ZEdUtil.Delete←Ch - exported

- ZEdUtil.InsDelBuf - exported

7.1.3.3.4.1 ZEdUtil.InsBlank

This procedure is used to implement the insertion of TAB characters into the current item at the current position by inserting the appropriate number of blank spaces.

- Get the "absolute" column position of the current position within the item, then use it to obtain a new position at the next largest position divisible evenly by 8 (the default number of spaces in a TAB)
- Fill in the difference from the current position to that new position with blanks with ZDspInc.DspInsChr and move the cursor with ZDspInc.DspCurR
- Update the current item's selection character, just in case

7.1.3.3.4.2 ZEdUtil.InsKcrKlf (internal procedure to Insert-Ch)

This is a special routine for inserting a carriage return or line feed into the text. This really means creating a new text item in the current item record.

- If the current item's bottom line is at the bottom of the frame, then Beep and exit
- Clear the current line from the current position to the end with ZDspInc.ClrLin
- Strip off the appropriate part of the text from the current item, and form a separate substring with it
- If the character was a <CR>, then prepend it to the string, otherwise, get a leading blanks count of the remaining string on the current line to maintain the indentation
- Invoke ZDspInc.DspInsLin to actually insert and display the "newline"
- Increment the current column by one, and display the the cursor there with ZDspInc.DspTxtPos

7.1.3.3.4.3 ZEdUtil.Insert-Ch

Insert a single character, with special cases for the control characters, <CR>, <LF>, Ctl-H, Ctl-L, Ctl-I, Ctl-W and Ctl-Q.

- If there is no text in the current item and the character is a control-H or control-L, just Beep and exit
- If there is no text in the current item, create a text record, insert and display the character in the newly created text, and exit
- If the new character is a <CR> or <LF>, then invoke InsKcrKlf
- Else, if the character is a control-H (backspace), do the appropriate backspacing algorithm, and delete the character in the string
- Else, if the character is a control-I (horizontal tab), insert the correct number of blanks

with ZEdUtil.InsBlank

- Else, if the character is a control-W, insert the date/time stamp with ZEdUtil.InsWhoWhen
- Else, if the character is a control-Q, do another read to get the next (quoted) character
- If the character is a <CR> or <LF>, invoke InsKcrKlf
- Else, invoke ZDspInc.DspInsChr to insert and display the new character, and move the cursor right one space

7.1.3.3.4.4 ZEdUtil.Insertion - exported

This is the upper level of the character insertion utilities. It reads a character from the user, then inserts and displays it in the appropriate place.

- Set the cursor to display a hollow block (with the special character for NULL in the modified PERQ standard font set), then read in a character from the user with ZCanvas.RdKeyEv
- Set up a loop to be exited when the user enters an escape (<INS>), or a mouse button click
- Check to make sure the character will fit if its a <CR>, then insert it in place with ZEdUtil.Insert - Ch;
- Turn the cursor back on, and read in the next character to continue the loop
- If a control-U was entered, "undo" the previous insertions with ZEdItem.CTL - U
- Return the cursor to its normal (solid block) mode

7.1.3.3.4.5 ZEdUtil.Delete - Ch - exported

This routine deletes the character at the current position in the current item.

- Initialize, and get the length of the current line of text
- If there's no next line AND there's no text in the current line or we're at the end of the current line, Beep and exit
- If we're at (past) the end of the current line, and there is a next line, then invoke ZDspInc.DspDelLin to do the deletion of the (implied) <CR>, and insert the <CR> into the delete buffer; else, signal that we're at the end of the current item.
- Else, if the character is not a Control H (backspace), then insert the character into the delete buffer
- Do the actual deletion, and redisplay of the modified line via ZDspInc.DspDelChr

7.1.3.3.4.6 ZEdUtil.InsDelBuf - exported

This is the routine used to insert deleted characters, one at a time, into the global delete buffer pointed to by BufP.

- Increment the active length of the buffer (i.e,. the character array) by one
- If there is room in the buffer, then store the character at the current position in the buffer
- Update the selection character of the current item, if necessary, via ZEdUtil.UpdSelCh

7.1.3.3.5 Item Initialization utilities

- ZEdUtil.CopyFsP - exported
- ZEdUtil.ItemInit - exported

7.1.3.3.5.1 ZEdUtil.CopyFsP - exported

This routine is called from ItemInit to copy the text of the selection pointed to by global SelPTyp. CurSelP, to a structure pointed to by the global FsPTyp, FirstStr.

- If there's no text in the current item, then just exit
- Initialize some of the working pointer variables
- For each line of text in the current selection, create a new string record with NetMakeDel.CrFsP and insert this record into the list. Make global FirstStr point to the head of the list

7.1.3.3.5.2 ZEdUtil.ItemInit - exported

This routine initializes the global data structure, CItem, exported from Module ZEdDefs, whenever the user moves to a new item in the frame.

- If the Current selection pointer, CurSelP is NIL, and the item [illegible] is either Text or Title, then create a selection for that type via NetMakeDel.CrSelF
- Copy the current selection text to a frame string record pointed to by global FirstStr via ZEdUtil.CopyFsP
- Store the current item's selection box and current position parameters in the global "original" variables, so the item can be reestablished if the user chooses to "undo" some changes
- Initialize the cursor to the beginning of the item, and make the connection between CItem and the current selection, and the current selection text pointed to by FirstStr, via ZDspInc.IniTxtPos

7.1.3.3.6 Item Updating utilities

- ZEdUtil.UpdateBox - exported
- ZEdUtil.BoxBoundary - exported
- ZEdUtil.Save←Chg - exported
- ZEdUtil.UpdSelCh - exported

7.1.3.3.6.1 ZEdUtil.UpdateBox - exported

This routine will modify the selection box parameters for the given item to account for text modifications within the item.

- Initialize, then count the number of lines in the current item, and get the length of the longest line within the item
- Set the upper left box parameters to be the X-Y position parameters of the item
- Set the right side of the selection box according to the maximum length of line in the item
- Set the bottom side of the selection box according to the number of lines in the item

7.1.3.3.6.2 ZEdUtil.BoxBoundary - exported

This routine updates the selection (touch) box of the Current item.

- Set the local string pointer to point to where global string pointer, FirstStr points, i.e., the first line of the current item
- Invoke ZEdUtil.UpdateBox with this local string pointer as an argument, to actually update the selection box parameters

7.1.3.3.6.3 ZEdUtil.Save – Chg - exported

This routine preserves changes made to an item by connecting the global working pointer, FirstStr, to the appropriate selection pointer in the actual frame being edited.

- If no changes were made, as determined by Global boolean, Changed, then release the records pointed to by FirstStr via NetMakeDel.RelFsP, set FirstStr to point to NIL, and exit
- Otherwise, set Global Frame change variable, XCHG, to true
- Update the selection box of the current item with ZEdUtil.BoxBoundary
- Release the text pointed to by the original current selection, then make it point to where FirstStr points
- Reset FirstStr to point to NIL

- Reset the Current Item string pointer to NIL
- Set Global current item change boolean, Changed, to false

7.1.3.3.6.4 ZEdUtil.UpdSelCh - exported

This routine will replace the current selection's selection character with the first character in the selection's text. It is most often used when changes have been made to a selection which MAY have resulted in the original selection character being deleted, or replaced.

- If there is no text record for the current item, just exit
- If there is text in the record, then store the first character of the first line of the text into the selection's selection character field
- Else, [illegible] a NULL character as the selection character

7.1.3.3.7 Item Positioning Utilities

- ZEdUtil.Move
- ZEdUtil.MoveR
- ZEdUtil.MoveL
- ZEdUtil.MoveU
- ZEdUtil.MoveD
- ZEdUtil.Drag
- ZEdUtil.GCorners
- ZEdUtil.GBoundary

7.1.3.3.7.1 ZEdUtil.Move

This routine "moves" the current item by first erasing it, incrementing its position parameters by the given amounts, and finally, redisplaying it.

- Turn the cursor off, and "erase" the item via ZEdUtil.Erase
- Update the current item's position parameters by adding the given increments to the current values
- Redisplay the current item with ZEdUtil.ReDsp, and turn the cursor back on

7.1.3.3.7.2 ZEdUtil.MoveR

This routine moves the current item one space to the right, if possible.

- Check to see if there's room at the right hand margin for the proposed move
- If so, then invoke ZEdUtil.Move with an increment of + 1 for the horizontal repositioning
- Else, just Beep

7.1.3.3.7.3 ZEdUtil.MoveL

This routine moves the current item one space to the left, if possible.

- Check to see if there's room at the left hand margin for the proposed move
- If so, then invoke ZEdUtil.Move with an increment of -1 for the horizontal repositioning
- Else, just Beep

7.1.3.3.7.4 ZEdUtil.MoveU

This routine moves the current item one up, if possible.

- Check to see if there's room at the top of the frame for the proposed move
- If so, then invoke ZEdUtil.Move with an increment of + 1 for the vertical repositioning
- Else, just Beep

7.1.3.3.7.5 ZEdUtil.MoveD

This routine moves the current item one down, if possible.

- Check to see if there's room at the bottom of the frame for the proposed move
- If so, then invoke ZEdUtil.Move with an increment of -1 for the vertical repositioning
- Else, just Beep

7.1.3.3.7.6 ZEdUtil.Drag

This is the utility procedure for moving an item around on the frame with the mouse pointer.

- Display Dragging message, and store the original row and column position of the current item, as well as the number of lines in the item and the number of column positions in the item (i.e., its height and width), and finally change the mouse cursor to the "dragging" cursor
- While the only mouse event is mouse movement, every time the position of the mouse cursor changes enough to be different from the previous original position and the movement is within the window do the following:
- Erase the item via ZEdUtil.Erase, update its row and column fields, and redisplay it via

ZEdUtil.RdDsp

- Update the stored original row and column information to correspond to its new position, and display the now row and column in the user display line
- Invoke ZCanvas.RdCanvas to get the next canvas event and update the mouse cursor's row and column position accordingly
- Reset the mouse pointer to its normal pointer

7.1.3.3.7.7 ZEdUtil.GetCorners

This routine is a utility for GPOS which obtains the four outermost corners of the rectangle defined by all the options on the current frame.

- Point to the first option in the option list
- Obtain and store each of the corners for that option's selection box
- For each of the remaining options, compare their corners with the saved corner values. If any of their corresponding values exceeds the saved ones, then replace. This guarantees that the final values obtained will in fact reflect the size of the "super" selection box defined by the conglomerated options.

7.1.3.3.7.8 ZEdUtil.GBoundary

This routine updates the position of each option in the current frame by the specified increments.

- Point to the first option in the frame's option list
- For each option in the list:
- Increment the row position, and the horizontal sides of the selection box by the specified horizontal increment
- Increment the column position, and the vertical sides of the selection box by the specified vertical increment
- Point to the next selection and repeat.

7.1.3.3.8 Miscellaneous utilities

- ZEdUtil.GetFldStrP - exported
- ZEdUtil.RemMark - exported
- ZEdUtil.MarkSel - exported
- ZEdUtil.FldCase - exported

7.1.3.3.8.1 ZEdUtil.GetRidStrP - exported

This routine removes null strings from a section of a frame string list.

- For each "line" in the frame string list, check to see if there is any text in the line. If not, then adjust the pointers in the list structure to point "around" the zero length text record, and save the memory allocated to the record via NetMakeDel.SavFsP
- Point to the next "line" in the list, and repeat until there are no more lines, or the specified end has been reached

7.1.3.3.8.2 ZEdUtil.RemMark - exported

This routine removes the "-" mark from the first line of the current selection text if there is either a next frame or an associated selection action.

- If there is text in the current item, and there is either action text or a valid next frame, then replace the "-" at character position 3 with a space (" "), and redisplay the cursor at that position. Display the " " via ZCanvas.PutCh, and ZIO.SendChar if there is an external display unit.

7.1.3.3.8.3 ZEdUtil.MarkSel - exported

This routine displays a "-" mark on the first line of the current selection text if there is neither a next frame nor an associated selection action.

- If there is text in the current item, AND there is neither action text nor a valid next frame, then replace the " " at character position 3 with a minus sign ("-"), and redisplay the cursor at that position. Display the "-" via ZCanvas.PutCh, and ZIO.SendChar if there is an external display unit.

7.1.3.3.8.4 ZEdUtil.FidCase - exported

This routine is used to convert the case of the letters of a user-entered frame id (i.e., the subnet name part of the frame id), to the standard as it appears in the subnet itself.

- Call NetString.PrsFid to parse off the subnet name from the frame id
- Invoke NetHandl.TSnDef, with the subnetname as parsed, to see if it is a valid subnet. A side result of TSnDef is to convert the subnet id to its proper case, as stored in file :zognet>Subnet.Index
- Use BaseLib.CvIntStr to convert the frame number returned by PrsFid to a string
- Concatenate the subnet id with the frame number string to return it

7.2. The Slot Editor, SLED

SLED is a special purpose editor within ZOG, created specifically for the entering of text at the end of pre-existing options, called Slots, in customized frames, called Environment frames. The motivating factor behind the design of SLED was to create a standard, easy-to-use mechanism for users to enter the arguments needed for agent execution. All agents have access to the text information entered into these slots as the parameters they need for operation. SLED is also used to edit specialized Task and AirPlan frames.

SLED makes use of pre-existing "slot" frames "behind" each slot option in the environment to provide it with information about the allowable values for entry into the slots themselves. The tools for accessing these "slot" frames are built into SLED.

7.2.1. Module ZSled

This module contains the main Sled procedure, SlEdFr, as well as a number of PRIVATE support routines, including a number of these which are AirPlan- specific. The module's overall construction is similar to that of ZED.

7.2.1.1 ZSled.InitSled

Although the Slot Editor, SLED, was initially created for editing Agent Environment frames, it also proved useful for editing AirPlan input frames within the AirPlan expert system environment, since it constrains the types of entries allowed in given *slots* in a frame. Thus, SLED initialization also includes initializing some AirPlan support data structures.

- Create a record for frames being edited via NetMakeDel.CrFP; this newly allocated record is pointed to by SledFP
- Set the local AirPlan boolean, AirPlanUp to false, indicating that AirPlan is not currently running
- Allocate memory for an AirPlan message data structure, pointed to by ZOGMsgP; This structure is defined in Module ZOGMsg
- Create a local pointer for AirPlan input messages as a recast of ZOGMsgP into an AirInPTyp variable, AirInP. AirInPTyp is defined in Module ZOGMsg.

7.2.1.2 Procedures SlEdFr, Editing and EnvStuff

Procedure SlEdFr is the entry point into SLED. It performs the initial setup for SLED, then invokes the main SLED Command Parser, Editing, in much the same way as ZED.

EnvStuff handles the actual editing of the selected slot in the environment frame.

7.2.1.2.1 ZSled.SlEdFr

This routine simply (re)displays the frame to be edited in the current window, invokes the initialization routine, ZSled.Init, and then invokes the SLED command parser, ZSled.Editing.

- Open the specified frame for editing, highlight any unfilled option slots via ZSled.InitHighLight and, if necessary, display it in the current window
- Invoke ZSled.Init to display the SLED messages and titles and reset the global editing variables to an initial state
- If the frame is an AirPlan input frame (as determined by the presence of "bred:" in the frame expansion area, invoke ZBREd.PreProc to split up the options
- Call ZSled.Editing, the SLED command parser to accept and interpret SLED commands from the user, until global boolean, Exiting, is set to true
- Invoke internal procedure, ZSled.SlEdFr.EndSledStat, to reset some of the display functions, and to record SLED ending times for Statistics record keeping

7.2.1.2.2 ZSled.Editing

This procedure is the top-level command interpreter/processor for SLED. All user commands entered from within SLED are interpreted through a CASE statement, with the appropriate routine being called to actually perform the SLED command.

- Display the cursor at the current item position, display the SLED message if necessary and get command character or mouse click via ZCanvas.RdKeyEv
- Begin command parsing loop by checking if a command was entered by the mouse, and if so, translating the mouse button click to a command character from the mouse's position on the bottom line (SLED "GPads")
- Parse the command character entered with a case statement, and invoke the appropriate procedure to execute the intended command. Only a very limited subset of commands are implemented in SLED, the "h", "e" and "q" commands, <LF> and <INS> and Control-U. The "D", "F" and "P" commands are only for AirPlan frame's deletion, reformating and repositioning.

7.2.1.2.3 ZSled.EnvStuff

This procedure invokes the actual slot editing functions contained in Module ZEnvEd. It is called from ZSled.LF, INS, Mouse, or FindSel, each of which routines ends up with global record, CItem, pointing to the option (slot) to be edited.

- Initialization - Set cursor/display characteristics, and insure that there is a space after the colon, if not AirPlan frame
- Editing - Invoke ZEnvEd.EdEnvFr to do the actual editing/slot entry

- Error Handling - Do a case statement on error number returned from EdEnvFr for proper error message display, then exit

- Redisplay new slot value - Do proper formatting, depending on whether the frame is an AirPlan frame or not, and redisplay the new slot value in the slot

- Wrap up - Reset cursor/display functions, set global "change" booleans, initialize option, redisplay SLED message, and send AirPlan packet if necessary

7.2.1.3 Sled Support Procedures

- ZSled.WhichItem
- ZSled.Mouse
- ZSled.LF
- ZSled.INS
- ZSled.Clean
- ZSled.Quit
- ZSled.OutSled
- ZSled.Init

7.2.1.3.1 ZSled.WhichItem

Given a row and column position of the mouse in the frame being edited, this procedure returns a pointer to the selection containing that row, column position, if it exists, returning a value of false in var parameter, FoundItem if there's no match.

- Set local selection pointer variable, ASel, to the first Option on the frame, and check with ZSled.FSelF. If successful, just exit.

- Set local selection pointer variable, ASel, to the first Local Pad on the frame, and check with ZSled.FSelF. If successful, just exit.

7.2.1.3.2 ZSled.Mouse

This procedure implements the mouse selection feature of SLED. If the user clicks the mouse within any option, this routine will find that option, and put the cursor at the slot position of that option for text entry.

- Get the row and column of the position of the mouse, and save the current item information in OSelP

- Invoke ZSled.WhichItem to determine the item inside of which the mouse was clicked

- If no match, give error message and exit
- Invoke ZSledUtil.Save - Chg to preserve any previous changes to the option
- Initialize globals to the found option with ZSledUtil.ItemInit and Start - item, and display the cursor at this starting position with ZDspInc.DspTxtPos
- Invoke ZSled.EnvStuff to do the actual editing of the slot for this option

7.2.1.3.3 ZSled.LF

This procedure implements the LineFeed (<LF>) command, which steps the user through the frame's options, starting at the current. and ending at the last local pad.

- If currently at last local pad in frame, BEEP and exit
- Save the current option pointer for later comparison to see if we've actually moved
- Save any changes made in the current item via ZSledUtil.Save - Chg
- Invoke ZSledUtil.LineFeed to do the actual stepping to the next option
- Place the cursor at the beginning of the new item, and initialize global pointers with ZSledUtil.ItemInit
- If, in fact a new option was reached, then invoke ZSled.EnvStuff to do the actual slot editing on the new option; else just Beep

7.2.1.3.4 ZSled.INS

This procedure implements the Escape (<INS>) command, which steps the user "up" through the frame's options, starting at the current option, and ending at the first option.

- If currently at frame title or text in frame, BEEP and exit
- Save the current option pointer for later comparison to see if we've actually moved
- Save any changes made in the current item via ZSledUtil.Save - Chg
- Invoke ZSledUtil.Escape to do the actual stepping to the previous option
- Place the cursor at the beginning of the new item, and initialize global pointers with ZSledUtil.ItemInit
- If, in fact a new option was reached, then invoke ZSled.EnvStuff to do the actual slot editing on the new option; else just Beep

7.2.1.3.5 ZSled.Clean

This procedure is used to remove the SLED messages from the display of the edited frame in the current window, prior to leaving SLED.

- Invoke ZSled.RmHighLight to erase any highlighting that was done
- Set the character display function to Replace mode via ZCanvas.SetChFunc
- Turn the cursor off, then position it at 1,59 in the frame, the position of the "*Sled*" string
- Display 6 spaces with ZCanvas.PutCh to "erase" that string
- Place the cursor at the user display line with ZDisplay.DspEndhen clear that line with ZDisplay.ClrLine
- Position cursor at the bottom with ZDisplay.DspEnd and clear it too
- Obtain and display the frame's global pads, via ZDisplay.DGPads
- Call ZCanvas.ChangeCanvas to redisplay the whole window

7.2.1.3.6 ZSled.Quit

Quit is used to implement the "q" command. The changes made to the frame are discarded, and the original frame is redisplayed.

- Invoke ZSledUtil.Save – Chg
- Erase the ZED messages and "global pads" with ZSled.Clean
- Set AirPlan boolean, Repack to false
- Do a quit for the currently open frame via NetHandl.QuitF
- If changes were made during the editing session, reread and redisplay the original unchanged frame via ZWind.RdFWind and ZWind.DspWind
- Set Module-wide boolean, Editing, to true

7.2.1.3.7 ZED ZSled.OutSled

OutSIED implements the Exit ("e") command. It is the routine which causes the edited copy of the frame to be written out to the subnet on the disk.

- Invoke ZSledUtil.Save – Chg
- Clean off the ZED related messages and strings on the frame being edited via ZSled.Clean

- If the current user is "GUEST" and changes were made to the frame being edited, give the user a message stating that no permanent changes are being made, and exit
- If the frame was an AirPlan input frame (RePack is true) then invoke ZBREd.PostProc to "pack" the options together
- If changes were made to the frame being edited, then close the frame, via NetHandl.ClsF, and read the new frame into the window structure with ZWind.RdFWind

7.2.1.3.8 ZSled.Init

This is the initialization routine called each time SLED starts up. It performs a variety of functions:

- Determine height of current window
- Load up Global String, EdMsg, with standard ZED user message
- Display message, "*Sled*" string, and new ZED title in window
- Initialize local and global variables, mostly booleans; Point to Text in current frame; Invoke ZEdUtil.ItemInit to initialize global variables CurSelP and CItem.
- Check if the user is "GUEST"; if so, notify user that no permanent changes will be made.
- Check to see if this is an AirPlan input frame; if so, then determine if automatic messages are to be displayed from the user.
- Remove and store old alternate cursor for duration of editing

7.2.1.4 Local Utilities

- ZSled.AppendHighlight
- ZSled.InitHighLight
- ZSled.RemoveHL
- ZSled.RmHighLight
- ZSled.CmpSelChr
- ZSled.FSelChr
- ZSled.FindSel
- ZSled.FSelF

7.2.1.4.1 ZSled.AppendHighlight

This routine fills in all slots in the passed selection list which have no default values in them with the string " ***", as an indication to the user that these slots must be filled in explicitly.

- Put the character display into XOR mode via ZCanvas.SetChFunc
- For each selection on the list, if the selection has text, then if the selection text ends with either a ":" or a space, append the string, " ***", to the end of the text, and display the string there

7.2.1.4.2 ZSled.InitHighLight

This routine merely invokes ZSled.AppendHighLight for both the frame's option list and local pad list, to insure that all slots are properly highlighted at SLED initialization time.

- Set the local selection pointer to the top of the option list
- Invoke ZSled.AppendHighLight with this pointer
- Set the local selection pointer to the top of the local pad list
- Invoke ZSled.AppendHighLight with this pointer

7.2.1.4.3 ZSled.RemoveHL

This routine removes the "highlights" (i.e., the string " ***") from the end of any selections (slots) on the passed selection list which weren't filled in by the user prior to closing the frame at Sled Exit time.

- Put the character display into XOR mode via ZCanvas.SetChFunc
- For each selection on the list, if there is text in the selection, then if the string " ***" is found in the selection's text, then overwrite it (in XOR mode to remove it from the display), and delete it from the text string via PERQ – String.Delete.

7.2.1.4.4 ZSled.RmHighLight

This routine merely invokes ZSled.RemoveHL for both the frame's option list and local pad list, to insure that all slots which were highlighted at SLED initialization time have their highlights removed.

- Set the local selection pointer to the top of the option list
- Invoke ZSled.RemoveHL with this pointer
- Set the local selection pointer to the top of the local pad list
- Invoke ZSled.RemoveHL with this pointer

7.2.1.4.5 ZSled.CmpSelChr

A lower-level routine to search through a list of selections, attempting to match each one's selection character with the character passed to it. A non- obvious, but utilized side effect of this routine is that it leaves global current selection pointer, CurSelP, pointing to either the matched selection or to the last selection on the list.

- Set booleans to false to initiate the search
- Starting with the current selection (CurSelP↑), check the each selection's selection character with the passed character parameter
- If they match, set var boolean, Found, to true to exit the loop
- Else, if there's a next selection, point to it for the next pass, else, set terminating boolean, EndGroup, to true to exit the loop.

7.2.1.4.6 ZSled.FSelChr

This is the middle-level selection character search routine.

- Store the current selection pointer, CurSelP, in a local variable
- If there are any options in the frame, set CurSelP to point to the top of the option list. Invoke ZSled.CmpSelChr to try to find a match. If it was successful, set Global variable TypeItem to be IOptions, and exit.
- If there are any local pads in the frame, set CurSelP to point to the top of the local pads list. Invoke ZSled.CmpSelChr to try to find a match. If it was successful, set Global variable TypeItem to be IPads, and exit.
- If none of the above searches were successful, Beep and restore CurSelP to point to its previous selection.

7.2.1.4.7 ZSled.FindSel

This routine implements the SLED "D", "F" or "P" (Find selection) command.

- If there are no options or local pads on the frame, beep and exit
- Save the current selection pointer
- Display the prompt for the selection character to match, and obtain it from the user
- Invoke ZSled.FSelChr to find the matching selection
- If the search was unsuccessful, Beep and exit
- Else save any changes that were made on the old current selection, set CurSelP to point to the new "found" selection, initialize the global variables for the new selection via ZSledUtil.ItemInit

- Invoke ZSled.EnvStuff to perform the actual editing of the selected option.

7.2.1.4.8 ZSled.FSelF

This procedure checks to see if the mouse position, passed to it via parameters row and col, falls within the selection box of any of the passed option type.

- Initialize var boolean, FoundItem, to false

- If the passed value of Col is greater than 99 (i.e., 100), then search through the option list, only checking the row positions of the options. If a match is found, set FoundItem to true, and exit. This mechanism is only used for AirPlan frames.

- Else, for every non-NIL value of ASel, the passed selection pointer, see if row is within the l0, l1 pair of the selection, and if col is within the c0, c1 pair of the selection; If so, set FoundItem to true, and exit; Else, get the next selection pointer, ASel↑.nextsel and repeat the checking.

7.2.1.5 AirPlan Support Local Utilities

ZSled.TestSN	Tests if S/N (Side Number) string is all digits
ZSled.ValueX100	Converts fuel value in hundreds of pounds to pounds
ZSled.GetHeading	Gets the AirPlan heading from frame expansion area
ZSled.GetAttribute	Gets the AirPlan attribute from current selection expansion area, if it exists
ZSled.GetObject	Gets the side number from an AirPlan option on specified row in frame
ZSled.SendAirMsg	Updates AirPlan input frame, forms AirPlan message from option info, and sends it across the Ethernet
ZSled.DelLine	Deletes an entire line (single aircraft) from frame when it has trapped
ZSled.RePos	Invokes "AirPlan" style repositioning of frame's options via ZSledUtil.Air-Repos

7.2.2. Environment Editing Support Modules

The procedures contained in these two modules provide the lower level interface between SLED and the environment mechanism, allowing SLED to access the information contained in the slot frames "behind" each option in the environment frame.

The procedures in ZEnvUtil are also exported to the agent environment frame accessing library, Module EnvLib for agents use in accessing their necessary parameters from the slots in their environment frames.

7.2.2.1 Module ZEnvEd

This module is the main workhorse of the Environmental Editor

7.2.2.1.1 ZEnvEd.EdEnvFr - exported

This is the top level routine for slot editing. It controls the user's input based on the slot attributes contained in the slot's (option's) slot frame.

- Initialize, and store the original slot value, if any
- Get the slot frame id for the slot (option) being edited with ZEnvUtil GSlotFrame
- Read in the slot frame into global frame record, FPX, witn NetHandl.RdF
- Get the slot's attributes from its slot frame via ZEnvUtil.GAttributes
- If the slot is a toggle/boolean type, handle it with ZEnvUtil.BoolToggle or ZEnvUtil.GToggle, respectively, and exit
- Otherwise, get the user's input for the slot either from a prompt frame or from a direct type-in through ZEnvUtil.GReply
- Verify the user's response against the slot type using ZEnvUtil.VReply

7.2.2.1.2 Slot Obtaining and Checking Utilities

- ZEnvEd.SelSearch
- ZEnvEd.GSlotValue
- ZEnvEd.SlotCheck
- ZEnvEd.GSlotFrame

7.2.2.1.2.1 ZEnvEd.SelSearch

This function searches through a selection list, attempting to match the given mask with the selection text. Returns an integer which points to the string to be matched in the found selection, and set var parameter, SelP, to point to the found selection.

- Initialize variables
- Search through the selection list, trying to match the given string with the selection text, if any
- If a match is found, exit the search loop, and set the function value to the index of the found string in the selection text

7.2.2.1.2.2 ZEnvEd.GSlotValue

This procedure returns the slot value from an entire slot text. It really only returns the first "word" of the slot value.

- Get the substring of the slot text from two positions after the ":" to the end
- Strip off anything from the string after and including the first space
- Invoke NetString.Strip to remove any extraneous leading or trailing blanks

7.2.2.1.2.3 ZEnvEd.SlotCheck

This function is used to determine if the frame currently in the FPX record is a slot frame, based on the occurrence of "Slot:" in the frame's title.

- Assume a return value of false
- If there's a title text in the frame, then if "Slot:" occurs in the title text return true

7.2.2.1.2.4 ZEnvEd.GSlotFrame

This routine will read in the slot frame into global FPX for later use. It also verifies that the frame is a slot frame, and process things properly if there is a "case" statement in the slot frame text. Errors are returned via coded error numbers.

- Obtain the slot frame id from the selection's next frame field, then from the expansion area (the preferred way - this should be an if-then-else statement on the existence of the expansion area)
- If there is a slotframe, read it into FPX via NetHandl.RdF
- If successful, then check to see if it is a valid slot frame with ZEnvEd.SlotCheck
- If successful, check to see if the slot frame has a case statement; If so, then get the case key string and the current value of the case slot via ZEnvEd.GCaseType and GCaseValue respectively. If both of these are successful, then invoke ZEnvEd.GCaseSlot to return the correct slot frame in global frame record, FPX.

7.2.2.1.3 Value Toggling Utilities

These procedures toggle the allowable slot values each time the slot (option) is selected within SLED.

- ZEnvEd.BoolToggle
- ZEnvEd.GToggleVal

7.2.2.1.3.1 ZEnvEd.BoolToggle

This routine will toggle the boolean values specified in the prompt option of the slot frame. These can be "yes" and "no", "true" or "false".

- Search the options of the slot frame contained in global record, FPX, for the occurrence of the string, "Prompt: "
- Obtain the text in that option following the ": ", as the values to be toggled
- Find the occurrence of the character ";", which is the separator between the two values to be toggled, and use its location to obtain the two separate toggling values
- Replace the old value of the slot with the "other" boolean value, and return the new value to the calling routine
- Note: the AirToggle boolean allows for the occurrence blank as a legal value for AirPlan frames

7.2.2.1.3.2 ZEnvEd.GToggleVal

This procedure will cycle through a "ring" of specified values for a given slot, starting with the value "after" the current value every time the slot (option) is selected while in SLED. The ring of specified values is stored as options in the prompt frame pointed to by the "Prompt: " option of the slot frame.

- Given the previous value of the slot, store it, then read in the prompt frame into global frame record, FPX
- Point to the top of the option list in the prompt frame, then search through the list for the option with a value which matches the old slot value
- If a match is found replace the slot value with the "next" option value from the prompt frame, if it exists and exit
- Else, replace the slot value with the value found as the text of the first option of the prompt frame

7.2.2.1.4 Case Utilities

These procedures implement the environmental editor's case facility, providing for different slot frames for a given slot, depending on the value of a different slot in the original environment frame.

- ZEnvEd.GCaseType
- ZEnvEd.GCaseValue
- ZEnvEd.GCaseSlot
- ZEnvEd.CaseCheck

7.2.2.1.4.1 ZEnvEd.GCaseType

This procedure returns the Case statement's "key" value, that is the "name" of the slot in the environment frame whose value is to be matched in implementing the case mechanism. This value is surrounded by double quotes.

- Point to the text in the Case Slot frame, contained in global FPX
- Search the text for the occurrence of a double quote, then for the second double quote
- Use these two indices to extract the substring of the frame text contained within these quotes

7.2.2.1.4.2 ZEnvEd.GCaseValue

This routine returns the value of the slot indicated by name in a slot frame's case statement.

- Point to the Environment frame's Option list
- Invoke ZEnvEd.SelSearch to find the slot (option) in this list which matches the value passed
- If none is found, do the same for the environment frame's local pads
- If a match is found, then invoke ZEnvEd.GSlotValue to extract the slot's "value", i.e., user-input string, to be returned

7.2.2.1.4.3 ZEnvEd.GCaseSlot

This procedure is at the heart of the case mechanism. Given a value from the proper slot, it goes through the option list of the "case" frame trying to find a match. If a match is found, the "next frame" is retrieved, and is the Slot frame to be used for the original slot. Note: this mechanism is inherently recursive, so nested case statements can be used.

- Use ZEnvEd.SelSearch to search the options of the slot frame stored in global frame record, FPX, for the occurence of the case slot's current value
- If none is found, use SelSearch to find an occurrence of "otherwise"
- If any match is made, invoke ZEnvEd.GSlotFrame (recursively) to obtain the corresponding slot frame to be returned and used

7.2.2.1.4.4 ZEnvEd.CaseCheck

This function is used to determine if the frame currently in the FPX record is a case frame, based on the occurrence of 'case "' in the frame's text.

- Assume a return value of false
- If there's frame text in the frame, then if 'case "' occurs in that text return true

7.2.2.1.5 Other Utilities

- ZEnvEd.GetOther
- ZEnvEd.GHelp

7.2.2.1.5.1 ZEnvEd.GetOther

This is a utility procedure which is only invoked when the user select an option labeled "Others" in a prompt frame. It allows the user to enter his/her own string.

- Display a prompt string in the user display line with ZSledUtil.WrtMsg
- Read the user's response with ZEnvUtil.RdReply

7.2.2.1.5.2 ZEnvEd.GHelp

This utility is to be invoked when the user selects SlEd command "h". It reads in and displays the help frame, if available. NOTE: This procedure is not invoked from anywhere! The help feature has not been implemented.

- Read in the Environment frame's slot's slot frame into FPX
- If successful, find the "Help" option on the slot frame with NetOption.GOptF, using "H" as the selection character (Note: this is not implemented)
- If found, display the help frame in the other window via ZWind.XChange, ZAAction.GoToFrame, ZWind.DspWind and finally, ZWind.XChange to get back to the original window
- Else, simply write out an error message in the user display line

7.2.2.2 Module ZEnvUtil

This module contains many of the utilities used by both the slot editor and the agent parameter obtaining mechanisms in agents Module EnvLib. It also exports the SLED-wide frame pointer, SledFP, which points to the frame currently being edited, as well as exporting Module ZDspIn from ZED.

7.2.2.2.1 Low level and specialized utilities

- ZEnvUtil.Fs+AnyPos - exported implementation of NetString.AnyPos for frame string records
- ZEnvUtil.WrtMsg - exported
- ZEnvUtil.CvFidCase
- ZEnvUtil.WhichItem
- ZEnvUtil.ProcessInput

- ZEnvUtil.MarkUnmark

- ZEnvUtil.GetMonth

- ZEnvUtil.DateCheck

7.2.2.2.1.1 ZEnvUtil.FsAnyPos - exported implementation of NetString AnyPos for frame string records

7.2.2.2.1.2 ZEnvUtil.WrtMsg

This routine displays the given string in the User Display line of the frame being edited.

- Turn off the cursor via ZCanvas.CursorOff

- Put the character display into replace mode via ZCanvas.SetChFunc

- Get and save the current cursor position with ZCanvas.GetPosCh

- Put the cursor at the User Display line with ZDisplay.DspEnd and clear that line with ZDisplay.ClrLine

- Invoke ZCanvas.PutStr to print out the string, then if there's an external display device, call ZIO.SendStr to output the string to it (not necessary in newer versions, since ZCanvas.PutStr handles that)

- Restore the cursor to it original position with ZCanvas.SetPosCh, return the display function to XOR mode and redisplay the cursor with ZCanvas.CursorOn

7.2.2.2.1.3 ZEnvUtil.CvFidCase

This routine is used to convert the case of the letters of a user-entered frame id (i.e., the subnet name part of the frame id), to the standard as it appears in the subnet itself.

- Call NetString.PrsFid to parse off the subnet name from the frame id

- Invoke NetHandl.TSnDef, with the subnetname as parsed, to see if it is a valid subnet. A side result of TSnDef is to convert the subnet id to its proper case, as stored in file :zognet>Subnet.Index

- Use BaseLib.CvIntStr to convert the frame number returned by PrsFid to a string

- Concatenate the subnet id with the frame number string to return it

7.2.2.2.1.4 ZEnvUtil.WhichItem

This procedure checks to see if the mouse position, passed to it via parameters row and column, falls within the selection box of any of the frame's options, starting with the one pointed to by parameter, ASel (SelPTyp).

- Initialize var boolean, FoundItem, to false

- For every non-NIL value of ASel, the passed selection pointer, see if row is within the l0, l1 pair of the option, and if column is within the c0, c1 pair of the option, i.e., if the mouse position passed to this routine via row and column is within the option's selection box.

- If so, set FoundItem to true, make var parameter, CurSelP point to the found option and exit

- Else, get the next selection pointer, ASel↑.nextsel and repeat the checking

7.2.2.2.1.5 ZEnvUtil.ProcessInput

This procedure returns a pointer to the selection in a prompt frame chosen by the user via a mouse button click or a selection character input.

- If it was a mouse button click the user entered, then determine the position of the mouse, and use those "coordinates" to find out which selection was chosen via ZEnvUtil.WhichItem

- Else use NetOption.GOptF/GPadF to return the selection based on the selection character the user entered

- If no match was found, then Beep

7.2.2.2.1.6 ZEnvUtil.MarkUnmark

This procedure is used to highlight a selected character (either with the mouse or via a space) with a block cursor (which is represented by the ASCII character corresponding to Octal 177 (delete)). The original font set of the PERQ had to be modified for this.

- Put the character display mode into XOR function with ZCanvas.SetChFunc

- Put the cursor at the specified row and column with ZDisplay.DspPos

- Display the block character with ZCanvas.PutCh

- Reset the character display mode to Replace function

7.2.2.2.1.7 ZEnvUtil.GetMonth

This routine is used to convert a user entered 3-character month string to proper case, as a utility for task frame editing.

- Set up the string of concatenated months with proper case

- Use NetString.AnyPos to do a case-insensitive search of the month string with the user-input month string

- If a match was found, then return the correct 3-character substring of the month string in the var string variable, NewMonth

7.2.2.2.1.8 ZEnvUtil.DateCheck

This routine is a utility to verify and, if appropriate, return a properly formatted, a user-input date for task frames. It essentially parses the user- input string, checking for a maximum 2-digit day, verifying the month with ZEnvUtil.GetMonth, and checking for a maximum 2-digit year. Once these checks have been successfully made, then it checks the day vis-a-vis the month to make sure it is a legal date.

7.2.2.2.2 Routines to obtain slot information

- ZEnvUtil.GSlot.FSlot
- ZEnvUtil.GSlot - exported
- ZEnvUtil.GAttributes - exported

7.2.2.2.2.1 ZEnvUtil.GSlot.FSlot

This internal procedure does the work for routine GSlot.

For each selection on the selection list, do:

7.2.2.2.2.1.1 Search for the occurence of the slot's name in the selection text

7.2.2.2.2.1.2 If found:

- Obtain the selection character, and the slot's "value", which is the string occurring after the colon in the selection text
- Obtain the slot frame id, which should be in the selection's expansion area, after the string "FrameId:", or use the selection's next frame if there's no expansion area
- Verify the Slot Frame Id with NetHandl.TFDef, and print an error message if the frame Id is not defined
- Exit GSlot

7.2.2.2.2.1.3 Point to the next selection

7.2.2.2.2.2 ZEnvUtil.GSlot - exported

This procedure returns the selection character, the (presumably user-filled-in) slot value and the frame id of the slot frame of the slot (i.e., option) whose "name" was supplied.

- Point to the frame's options, and invoke internal procedure, FSlot, to do the work
- Point to the frame's local pads, and invoke internal procedure, FSlot, to do the work
- Since FSlot will exit GSlot (!) if values are found, then if we're still in GSlot at this point,

print an error message, and return false

7.2.2.2.2.3 ZEnvUtil.GAttributes - exported

This routine parses the slot frame for a given slot, and returns the attributes of that slot, including type, default value, prompt information, and the frame id's of the prompt and help frames, if they exist. The slot frame has already been read into the global FPX frame record.

- Initialize var parameters, then step through the slot frame's options
- Based on each option's text, obtain the Slot Name, Slot Type, Slot's Default value, Slot Prompt information, and Slot Help information
- Verify Slot Type information
- Form default prompt, if necessary
- Set Var boolean, Sig, to true

7.2.2.2.3 Routines to obtain values from user

- ZEnvUtil.RdReply - exported
- ZEnvUtil.GReply - exported
- ZEnvUtil.VReply - exported
- ZEnvUtil.REnvVal - exported

7.2.2.2.3.1 ZEnvUtil.RdReply - exported

This procedure provides for the entering of slot values, after the slot name. It handles the BackSpace (delete) character properly. The terminating character is an escape (INS), <CR>, or <LF>. It returns the user-input string.

- Position the cursor at the specified position (after the slot name)
- Read in a character via ZCanvas.RdKeyEv
- Enter a loop to parse the character, enter it into the var string variable, str, and obtain the next character, until a terminating character is received

7.2.2.2.3.2 ZEnvUtil.GReply - exported

This is the main routine to obtain slot values from the user using the menu provided in a prompt frame.

7.2.2.2.3.2.1 If there is a prompt frame then

- Read in the prompt frame into global frame record, FPX. If unsuccessful, display error message, and exit
- If parameter NoPrompt (for AirPlan input frames) is false, then display the prompt message, switch windows, and display the prompt frame there, else, just display the AirPlan prompt frame message
- Read a character from the user; if it is an escape, then use the value already in the slot, else use ZEnvUtil.ProcessInput to obtain the selection the user specified from the prompt frame
- If parameter NoPrompt is false, then redisplay the original frame in the other window and return to current window

7.2.2.2.3.2.2 Else

7.2.2.2.3.2.2.1 Initialize, then enter repeat loop to obtain a response from user

- Point to after the colon at the end of the slot name
- Construct the prompt message, doing special things for task frames
- Display the prompt message, and obtain the currently display value of the slot
- Locate input position, then invoke ZEnvUtil.RdReply to obtain the user-entered string
- If the length of the string is zero, then use either the default value from the prompt frame, or the previously displayed string
- If a non-zero reply has been obtained, terminate the loop

7.2.2.2.3.2.2.2 If there was a leading blank in the user string, eliminate it an any other extraneous blanks with NetString.Strip

7.2.2.2.3.3 ZEnvUtil.VReply - exported

This procedure verifies the user-entered slot value against the type for the slot as specified in the slot frame, after coverting abbreviations. It is essentially one large case statement on the first letter of the SlotType String.

B	Boolean - Verify that response is in ['f', 'n', 't', 'f']
C	Character - Verify that response is only one character long
I	Integer - Verify that all the characters are digits
P	PString - (No Checks are made, since none are meaningful)
R	Real - Verify that all characters are in ['-', '0'..'9', '.']

F	FrameId - Convert '.' or '/' to a frameid string and verify that the string refers to a valid, defined frame
S	SubnetId - Convert '.' or '/' to a subnetid string and verify that the string refers to a valid, defined subnet
D	Date (for task Frame) - Verify the date string with ZEnvUtil.DateCheck
T	Time (for task Frame) - Verify that the time is valid
U	UserId - Verify that the user id is valid with internal procedure, VUserId

7.2.2.2.3.4 ZEnvUtil.REnvVal - exported

This routine was used to redisplay the environment frame with the slot values properly formatted prior to rewriting the frame out to disk. It is no longer used.

7.2.3. Utilities Modules

These modules provide lower level support for the Slot Editor. Module ZSledUtil, in particular, provides the same kind of support to SLED that Module ZEdUtil provides to ZED. Module ZBRED, on the other hand, provides specialized support for editing specific types of AirPlan input frames.

7.2.3.1 Module ZSledUtil

This module contains many of the same routines contained by modulo ZEdUtil, most modified slightly to fit in with SLED's requirements. It also imports Module ZEdDefs to have local access to all of ZED's global variables for its own use.

- Display utilities
- Item initialization utilities
- Item updating utilities
- Command Implementing procedures
- AirPlan Editing Utilities

7.2.3.1.1 Display utilities

- ZSledUtil.BEEP - exported - Invoke ZCanvas.BEEPCanvas to sound the "bell" [illegible] the default SLED message.
- ZSledUtil.DspFPEd - exported
- ZSledUtil.DspEditorHdandTail - exported
- ZSledUtil.Erase
- ZSledUtil.ReDsp

7.2.3.1.1.1 ZSledUtil.BEEP - exported - Invoke ZCanvas.BEEPCanvas to sound the "bell" and display the default SLED message.

7.2.3.1.1.2 ZSledUtil.DspFPEd - exported

This procedure simply displays the frame stored in the editor frame record pointed to by EdFP.

- Set global SigNoClear to false
- Call ZDisplay.DspF to display the frame body
- Call ZDisplay.DGPads to display the associated global pads of the frame

7.2.3.1.1.3 ZSledUtil.DspEditorHdandTail - exported

This routine displays the "*SLED*" string in the upper right corner of the edit frame window, and the SLED "global pads" on the bottom line of the window.

- Put the character display in replace mode, via ZCanvas.SetChFunc
- Place the cursor at the User Display Line with ZDisplay.DspEnd, and clear it with ZDisplay.ClrEoLn
- Use ZCanvas.PutStr to display the Default Sled message
- Place the cursor at 1,59 via ZDisplay.DspPos and display "*SLED*" via ZCanvas.PutStr
- Position the cursor at the beginning of the bottom line with ZDspInc.DspPos, and clear that line with ZDisplay.ClrEoLn
- Form the "global pads" string and display it with ZCanvas.PutStr
- Restore the character display mode to XOR function

7.2.3.1.1.4 ZSledUtil.Erase

This is a general routine to clear the current item.

- Invoke ZSledUtil.Start – Item to set the current item variables to point to the beginning of the item
- For each line in the item, display the cursor at the beginning of the line, invoke ZDspInc.ClrLin, and point to the next line
- Reposition things at the beginning of the item via ZSledUtil.Start – Item

7.2.3.1.1.5 ZSledUtil.ReDsp

This is a general routine to (re) display the current item.

- Invoke ZSledUtil.Start – Item to set the current item variables to point to the beginning of the item
- For each line in the item, display the cursor at the beginning of the line, invoke ZDspInc.DspLin, and point to the next line
- Reposition things at the beginning of the item via ZSledUtil.Start – Item

7.2.3.1.2 Item initialization utilities

- ZSledUtil.Start – Item - exported
- ZSledUtil.CopyFsP
- ZSledUtil.ItemInit - exported

7.2.3.1.2.1 ZSledUtil.Start – Item - exported

This is used as a utility to initialize the cursor to the beginning of the current item.

- Set the current item's current relative row and column position to 1
- Make the current item's current string pointer point to the first string of the item
- Display the cursor at the (new) current position within the current item with ZDspInc.DspTxtPos

7.2.3.1.2.2 ZSledUtil.CopyFsP

This routine is called from ItemInit to copy the text of the selection pointed to by global SelPTyp, CurSelP, to a structure pointed to by the global FsPTyp, FirstStr.

- If there's no text in the current item, then just exit
- Initialize some of the working pointer variables
- For each line of text in the current selection, create a new string record with NetMakeDel.CrFsP, and insert this record into the list. Make global FirstStr point to the head of the list.

7.2.3.1.2.3 ZSledUtil.ItemInit - exported

This routine initializes the global data structure, CItem, exported from Module ZEdDefs, whenever the user moves to a new item in the frame.

- Copy the current selection text to a frame string record pointed to by global FirstStr via ZEdUtil.CopyFsP

- Store the current item's selection box and current position parameters in the global "original" variables, so the item can be reestablished if the user chooses to "undo" some changes.

- Initialize the cursor to the beginning of the item, and make the connection between CItem and the current selection, and the current selection text pointed to by FirstStr, via ZDspInc.IniTxtPos

- Redisplay the cursor at the current position via ZDspInc.DspTxtPos

7.2.3.1.3 Item updating utilities

- ZSledUtil.UpdateBox - exported

- ZSledUtil.BoxBoundary - exported

- ZSledUtil.Save-Chg

7.2.3.1.3.1 ZSledUtil.UpdateBox - exported

This routine will modify the selection box parameters for the given item to account for text modifications within the item.

- Initialize, then count the number of lines in the current item, and get the length of the longest line within the item

- Set the upper left box parameters to be the X-Y position parameters of the item

- Set the right side of the selection box according to the maximum length of line in the item

- Set the bottom side of the selection box according to the number of lines in the item

7.2.3.1.3.2 ZSledUtil.BoxBoundary - exported

This routine updates the selection (touch) box of the Current item.

- Set the local string pointer to point to where global string pointer, FirstStr points, i.e., the first line of the current option

- Invoke ZSledUtil.UpdateBox with this local string pointer as an argument, to actually update the selection box parameters

7.2.3.1.3.3 ZSledUtil.Save-Chg

This routine preserves changes made to a option by connecting the global working pointer, FirstStr, to the appropriate selection pointer in the actual frame being edited.

- If no changes were made, as determined by Global boolean, Changed, then release the records pointed to by FirstStr via NetMakeDel.RelFsP, set FirstStr to point to NIL, and exit

- Otherwise, set Global Frame change variable, XCHG, to true

- Update the selection box of the current item with ZSledUtil.BoxBoundary

- Release the text pointed to by the original current selection, then make it point to where FirstStr points

- Reset FirstStr to point to NIL

- Reset the Current Item string pointer to NIL

7.2.3.1.4 Command Implementing procedures

- ZSledUtil.CTL-U - exported

- ZSledUtil.LineFeed - exported

- ZSledUtil.Escape - exported

7.2.3.1.4.1 ZSledUtil.CTL - U - exported

This routine implements the "<Control-U> | <OOPS-key>" command, to "undo" the last changes to the current item.

- If there is no text in the current item (i.e., current row is row 1, and there is no text on this line), then just Beep and exit

- Erase the item via ZSledUtil.Erase (which simply overwrites the item with itself in XOR mode)

- If there were any changes made to the item (Global boolean changed is true), then release the top of the string list via NetMakeDel.RelFsp, and reset the current selection record position and selection box items to their original values (which were saved via a previous ZSledUtil.ItemInit call!)

- Redisplay the original current item via ZSledUtil.ReDsp

7.2.3.1.4.2 ZSledUtil.LineFeed - exported

This procedure implements the "<LF>" (move "down" to next item) command. It operates on a case statement by the type of current item:

TypeItem = lPads Set the Current Selection pointer, CurSelP, to the point to the next local pad

TypeItem = lOptions
Set the Current Selection pointer, CurSelP, to point to the next option. If there isn't one, then set it to point to the first local pad and adjust Global TypeItem accordingly

TypeItem = lText Set the Current Selection pointer, CurSelP, to point to the first option and adjust Global TypeItem, if there is one. If not, then make it point to the first local pad with the appropriate change to Global TypeItem, if there is one.

7.2.3.1.4.3 ZSledUtil.Escape - exported

This procedure implements the "escape" (<INS>) (move "up" to previous item) command. It does so via a case statement based on the current TypeItem:

TypeItem = IOptions
Set the Current Selection pointer, CurSelP, to the previous option, if one exists

TypeItem = IPads Set the Current Selection pointer, CurSelP, to point to the previous local pad, if there is one. If not, then if there are any options, make it point to the last option by starting at the first option and stepping through them to the last one, then updating Global TypeItem to IOptions.

7.2.3.1.5 AirPlan Editing Utilities

- ZSledUtil.InfoSwap
- ZSledUtil.DspAsel
- ZSledUtil.Del – AirPlan – Style - exported
- ZSledUtil.Air – Reformat - exported
- ZSledUtil.RstExp
- ZSledUtil.Clevage
- ZSledUtil.RealMove
- ZSledUtil.DoMove
- ZSledUtil.Air – Repos - exported

7.2.3.2 Module ZBRED

This module, so named for the *BackRoom Editor*, contains procedures which temporarily split up a single option in an AirPlan input frame into up to 10 distinct options for SLED type editing. This break up is based on the header string of the frame, contained in the frame text. At the close of the editing session, the options on a single line are *put back together* into a single option. This process is necessary because there is not enough space allocated on disk for a single *big* frame with a sufficient number of options to allow them to be edited for AirPlan input.

- ZBRED.Ge.SelExp
- ZBRED.SetBoxus
- ZBRED.CntFields
- ZBRED.BreakUp

- ZBRED.PutBackTogether
- ZBRED.PreProc - exported
- ZBRED.PostProc - exported

7.2.3.2.1 ZBRED.GetSelExp

This routine inserts lines of text from the frame's expansion area into option expansion areas, one line per option. This is done to load up default values for options.

- Search the frame expansion text for the occurrence of "******" and point to the next "line" of expansion text when found
- Starting at the first option, create an expansion area for the option with NetMakeDel.CrFsP, using successive lines of frame expansion text
- Do the above for each option until there are either no more options or no more frame expansion text

7.2.3.2.2 ZBRED.SetBoxes

This procedure resets the size of an option's selection box to be one line high and just as long as its text.

- Set the left edge of the selection box to the option's starting column position
- Set the right edge of the selection box to the end of the option's text
- Set both the top and bottom edge of the selection box to the option's row

7.2.3.2.3 ZBRED.CntFields

This procedure counts the number of separate AirPlan options in a single option (i.e., line) of the AirPlan frame based on the structure of the option header line contained as the frame text. This information is returned in Module-wide record variable, Field.

- Parse the header string for blanks
- For each blank in the header string, update the Field record to show the updated number of AirPlan options, the starting position of the new option (equal to the position of the blank + 1), and the length of the new option.

7.2.3.2.4 ZBRED.BreakUp

This is the procedure which does the actual breaking up of a single option (i.e., one line on the original AirPlan input frame) into the correct number of options, as contained in Module-wide record, Field.

For each new option to be established from the original option, do:

- Obtain the correct portion of the text for the new option
- Set its selection character to be the number of the option
- Create the new option with NetMakeDel.CrSelF
- Insert the new option into the correct place in the frame's option list
- Update its selection box with ZBRED.SetBoxes
- Delete that portion of text from the original option's text

7.2.3.2.5 ZBRED.PutBackTogether

This procedure will convert one line of options from an AirPlan input frame into a single option for writing out to disk.

For each row of options in the AirPlan input frame, do:

- Concatenate all the text of the options on that line into a single line of text
- Create a new string pointer with NetMakeDel.CrFsP with the new option string
- Create a new option record with NetMakeDel.CrSelF, with the first character of the option string as a selectio character
- Insert the new option into the correct place in the frame option list
- Update the selection box of the new option with ZBREd.SetBoxes

7.2.3.2.6 ZBRED.PreProc - exported

This procedure is called from ZSIEd.SIEdFr if the frame's options are to be temporarily split apart for editing.

7.2.3.2.6.1 First, make sure this frame's options are to be broken apart; if not, exit with var boolean, Success, set to false

7.2.3.2.6.2 Count the number of options to be created per line via ZBRED.CntFields

7.2.3.2.6.3 For each option (i.e., line) in the original frame, do:

- Invoke ZBREd.BreakUp to create the new options from the original
- Update the frame's option list to incorporate the new options
- If we're at the first option in the original option list, then update the new options'

expansion areas with ZBREd.GetSelExp;

7.2.3.2.6.4 Finally, release the original selection record of the frame, which contained the original option structure, via NetMakeDel.RelSelP

7.2.3.2.7 ZBRED.PostProc - exported

This procedure is called from ZSIEd.OutSIEd to recreate the original option structure of the frame.

- For each option in the frame, eliminate the option's expansion area, if any
- Point to the first option
- Invoke ZBREd.PutBackTogether to *concatenate* each line of options into a single option
- Make the frame's Option pointer point to the first option of the new option list
- Release the original option list via NetMakeDel.RelSelP

7.2.4. Environment Mechanism:

The mechanism by which the slots in an environment frame are controlled is central to the operation of both the Slot Editor and the agent parameter- obtaining mechanism. The information in the slot frame controls the allowable values of the user's entries, as well as data type. In addition, prompts for the user are supplied therein, including menu types of prompts for specially restricted values for the slot.

An example slot, representative of the slots found on "real" environment frames, is shown below, with a representative slot frame behind it. Normally, the slot frame is accessed via the "hidden next frame" mechanism, but in this case, it is through the normal vehicle.

7.2.4.1 Slot: Sample Slot:

This is a representative slot frame. Follow the options to obtain more detailed information about them.

- **Name: Sample Slot**
- **Type: String**
- **Default: Sample Information**
- **Prompt: Enter any information you care to, up to 80 characters**
- **Help:**

7.2.4.1.1 Name:

The "Name" option on the slot frame merely provides a check against the original option in the environment frame. The "name" simply means the text contained as the prompt in the environment frame's slot option.

7.2.4.1.2 Type:

The "Type" option specifies the data type allowed in the environment frame slot. Type checking is carried out via Sled to help the user enter the correct type of information. The following types are currently implemented:

PERQ PASCAL Types ZOG Types AirPlan Types --------------- -------- ------------

- Boolean
- Character
- Integer
- PString (String[256])
- Real
- FrameId
- SubnetId
- Boolean Toggle
- Date
- Time
- UserId

7.2.4.1.3 Default:

The "Default" option simply provides the default value for the environment frame slot to be used by the agent when the user enters no information therein. The default MUST be of the type specified by the "Type" option.

Several abbreviations are in current use. These are the "." and the "/":

- The "." stands for the current <item>, where <item> can be a Frame Id, Subnet Id, User Id, time or date. In the case of Frame or Subnet Ids, it refers to the frame displayed in the current window.

- The "/" is only used for Frame and Subnet Ids; it refers to the frame currently displayed in the other window.

7.2.4.1.4 Prompt:

The "Prompt" option on the slot frame serves two purposes. If there is NO next frame, then the text after the "Prompt: " will be displayed in the user display line as a prompt for the user when he/she selects the corresponding option on the environment frame.

However, if there IS a next frame, then that frame will be (temporarily) displayed in the OTHER window, which then becomes the current window. The user is directly to choose one of the options on that frame. The prompt frame displayed is typically a set of choices which the user is constrained to make by typing the selection character, or using the mouse.

7.2.4.1.5 Help:

The purpose of the "Help" option was to provide additional help information to the user for the particular slot he/she is editing. Either the text after the "Help: " will be displayed, or the frame pointed to by this option will be displayed in the other window.

8. Agents

There are a multitude of agents in the ZOG system, whi [illegible] provide a wide variety of functions (applications) for processing ZOG subnets (or portions thereof). These fall into the following categories:

- Planning and Evaluation (Task Management) Agents
- Backup and Tra[illegible] Agents
- ZOG Special [illegible] Agents
- Subnet Repair and Updating Agents
- SORM and Weapons Elevator Agents

8.1. Planning and Evaluation (Task Management) Agents

AgAdjDt	Adjusts the dates and times in a specific task tree
AgGenr	Creates a generic task tree from a specific task tree
AgGreen	Submit task to Green Sheet
AgInst	Instantiates a specific task tree from a generic task tree
AgInTask	Initializes a specific task tree
AgTPlan	Creates a task plan from a specific task tree in disk file form for outputting to a hard copy device
AgUpTask	Updates a task tree "upward" to propagate leaf node changes
AgZPlan	Creates a task plan from a specific task tree in a new tree of ZOG frames
AgPlan	Creates a task plan from a specific task tree in disk file form for outputting to a hard copy device

8.2. Backup and Transport Agents

These agents are used by system maintainers for reformatting subnets for backup and transportation

AgArchive	Archive a subnet or frame to a floppy
AgBackup	Write zbh for all perqs
AgBak	Write zbh for all subnets modified since a specific date and time for a specified Perq
AgVBH	Write Perq ZOG frames in VAX zbh format
AgZBH	Write zbh format of Perq ZOG frames

8.3. ZOG Special function Agents

8.3.1. Writing frames in a form suitable for printing

AgDoc	Write a tree of frames into a disk file using a format suitable for printing
AgPic	Write a single frame into a disk file without changing the format

8.3.2. Saving old versions of frames

AgOld	Copies a frame, linking the copy to the frame through an Old local pad
AgPost	Saves the current version of a frame as an Old frame, then clear the frame ,next copy the schema of the 0th frame to the current frame

8.3.3. Utilities

AgHiSubNum	Vinson utility routines
AgLink	Links an option to the frame in the other windo in an accessor-like manner. (Experimental)
AgMessage	Send a message to another Perq
AgTest	Schema for creating new agents
AgCode	Create a text file ready for compiling from a code subnet

8.3.4. Fonts and Graphics

AgBar	Creates a bar graph from a given data frame
AgRFont	Changes the fonts for a given subnet

8.3.5. Creating an index or directory of subnets

AgAlphaSNL	Creates an index of subnets either alphabetically or by Perq
AgDir	Creates a directory for subnets on a Perq or all Perqs
AgIndex	Creates an alphabetical index to subnets on a Perq or all Perqs

8.4. Subnet Repair and Updating Agents

AgMerge	Standardizes a subnets local pads to that of a given schema frame
AgPar	Corrects all bad parent and top links
AgProt	Modifies the protection on frames
AgSwap	Global string replacement
AgOwn	Adds or deletes the owner of a frame or frames
AgChkSecond	Checks secondary copy of a subne

8.5. SORM and Weapons Elevator Agents

The SORM and Weapons Elevator Agents are very specialized. Most of the following agents are used in formatting the document that is produced when the SORM and Weapons Elevator subnets are written out.

8.5.1. AgDgm : Writes out a chapter of diagrams

This agent will print, in scribe compatable format, a tree of frames. The format is for diagrams and GAPL(Government Allowance Parts List). Each frame corresponds to one picture and each picture may have a GAPL associated with it.

8.5.2. AgGAPL : Prints a tree of frames in scribe compatable format

This agent will print in scribe compatable format, a tree of frames. The format is for GAPL(Government Allowance Parts List) and prints a depth first search list of all parts in the tree.

8.5.3. AgMgmt : Produces a listing of all the frames title text

This agent will produce a depth first listing of all the frames title text in addition to a cross reference to the current frame. It is currently being used to generate the Apendix for the management codes in the *organization* section of the ships SORM.

8.5.4. AgOpr : Prints a tree of fraems in depth first search order

This agent will print a tree of frames in depth first search order. It is intended to print the *operate* section of the Weapons Elevator Manual. It's main features are that it prints out title text and a mini table of contents for each frame that has options.

8.5.5. AgOrg : Prints lists of responsibilities of each billet

This agent will print a tree of frames in depth first search order. It is intended to print the *organization* section of the ship's SORM. It prints a list of responsibilities of each billet with a cross reference into the task net where it is defined.

8.5.6. AgTask : Prints out option text for each frame that has options

This agent will print a tree of frames in depth first search order. It is intended to print the *understand* section of the Weapons Elevator Manual and the *operate and maintain* section of the ship's SORM. It's main features are that it prints out option text for each frame that has options. It will also print a mini table of contents if the frame has the keyword "CONTENTS" in the frame comment area.

8.5.7. AgText : Prints out the frame text on each frame visited

This agent will print a tree of frames in depth first search order. It simply prints the frame text on each frame visited. In addition it will follow any ">.More" local pads and follow any option tree that exits on the "More" frame.

8.5.8. AgThy : Prints out theory section of Weapons Elevator Manual

This agent wil print a tree of frames in depth first search order. It is intended to print out the *theory* section of the Weapons Elevator Manual and the ship's SORM. Option text is printed as the first sentence of a paragraph with any frame text on the next frame appended to the end. Each succeeding level is treated as a subparagraph of the proceeding paragraph and is indented as in an outline. No local pad cross references are generated in this agent.

8.5.9. AgTrb : Prints out troubleshooting section of Weapons Elevator Manual

This agent will print out a tree of frames in depth first search order. It is intended to print the *troubleshooting* section of the Weapons Elevator Manual. It's main features are that it prints out title text and a mini table of contents for each frame that has options. It also generates "subchapter", "section", "subsection", and "paragraph" commands for the first 4 levels in the tree. I ich successive level has the title text printed in bold face type.

8.5.10. AuxOrg : Prints out the appendixs for the ship's SORM

This agent will print out a tree of frames in depth first search order. It is intended to print the appendixs in the ship's SORM for parts of the organization such as department heads, division officers, leading chiefs, etc. Its only difference from AgOrg is that it will first mark a tree of frames as having already been seen so that duplication will be avoided when only a partial list is desired. As a side effect of having seen a frame before a cross reference is generated. In this way the list in an appendix of division officers will point to the correct location in the organization chapter. This is really somewhat of a kludge since we cannot keep the frames vid from running AgOrg around for a seco visited from running AgOrg around for a second run.

INITIAL DISTRIBUTION

Copies	
1	USS CARL VINSON
1	ONR/270
12	DTIC

CENTER DISTRIBUTION

Copies	Code	Name
1	18	G. Gleissner
1	1808	D. Wildy
1	182	A. Camara
1	1826	J. Garner
1	1826	J. Jeffers
10	1826	D. Schmelter
1	522.1	TIC (C)
1	522.2	TIC (A)
1	93	L. Marsh